Distributed Consensus, Revisited *

Rachele Fuzzati
EPFL
Switzerland

Massimo Merro
Dipartimento di Informatica
Università di Verona, Italy

Uwe Nestmann
Technical University Berlin
Germany

Abstract

We provide a novel model to formalize a well-known algorithm, by Chandra and Toueg, that solves *Consensus* among *asynchronous distributed processes* in the presence of a particular class of *failure detectors* ($\diamond\mathcal{S}$ or, equivalently, Ω), under the hypothesis that only a minority of processes may crash. The model is defined as a global transition system that is unambigously generated by local transition rules. The model is syntax-free in that it does not refer to any form of programming language or pseudo code. We use our model to *formally* prove that the algorithm is correct.

1 Introduction

In the field of Distributed Algorithms, a widely-used computation model is based on asynchronous communication between a fixed number n of connected processes. No timing assumptions are made, neither on communications nor on local actions of processes. Often, processes are assumed to be subject to crash-failure: once crashed, they do not recover.

In this paper, we focus on distributed programming and coordination problems in the area of asynchronous models. In particular we are interested in (1) how the problems are typically specified, (2) how algorithmic solutions to such problems are described, and (3) how the solutions are shown to be correct with respect to their specification. In our opinion, specifications, solutions and correctness arguments are often presented at a too informal level. The offered amount of detail is not sufficient to fully convince the reader (especially an outsider to the field) of the validity of the arguments: a particular reader who wants to verify the correctness of some proofs often has to prove by herself substantial parts or entire sub-results, for which only informal arguments were given. In contrast, at the core of this paper, we propose a rigorous method to formally describe problems, algorithmic solutions and their respective correctness proofs at a fine-grained level of detail. The method builds upon a largely syntax-free modeling of algorithms as executable transitions systems. It is exemplified on the well-known problem of *Distributed Consensus* (or shortly: *Consensus*).

Specification. Usually, distributed programming problems are specified in terms of (often temporal) properties of admissible executions. Such an execution, also called *system run*, represents a (potentially infinite) computation, starting from some initial state, and describing the global behavior of a system as a sequence of actions and configurations according to some discrete time-line. An algorithm is an artifact that generates system runs. Often, some characteristics of components are given with respect to actions that do or do not happen in system runs. For example, a process is called *correct* in a given run, if it does not crash in that run. A solution to a problem is an algorithm that only generates system runs that respect the required properties. In the case of Consensus, a correct algorithm should only originate system runs that satisfy the following three properties:

*The original publication is available at www.springerlink.com

1. If a process decides a value v, then v was proposed by some process (*Validity*).
2. No two correct processes decide differently (*Agreement*).
3. Every correct process (eventually) decides some value (*Termination*).

In a seminal paper [FLP85], Fischer, Lynch and Paterson stated that, in the aforementioned asynchronous crash-failure model, it is impossible to solve Consensus—i.e., it is impossible to simultaneously satisfy the three properties above—if some process may fail. More precisely, they prove that, under the assumption of Validity and Agreement, the possibility of even a single process crash invalidates Termination: there would always be a non-terminating system run, where no decision is reached. Thus, while the properties of Validity and Agreement can be guaranteed through appropriate algorithms, in order to respect Termination it is necessary to introduce some notion of *synchrony* into the system.

A Solution by Chandra and Toueg. Since the discovery of the impossibility result of [FLP85], several refinements of the purely asynchronous computation model have been developed, also in order to make Consensus solvable. One of them is the addition of *unreliable failure detectors* (FDs), i.e., modules that can be locally queried to find out whether another process is currently locally suspected to have crashed [CT96, CHT96]. FDs are unreliable in that they may have wrong suspicions, they may disagree among themselves, and they may change their suspicions at any time. To become useful, the behavior of FDs is constrained by abstract reliability properties about (i) the guaranteed suspicion of crashed processes, and (ii) the guaranteed non-suspicion of correct processes. Both kinds of constraints are again expressed as properties on the set of system runs. For example, the failure detector $\diamond\mathcal{S}$ guarantees the following two properties on system runs: (1) *Eventual Weak Accuracy*: there is a time after which *some* correct process is never suspected by *any* correct process; (2) *Strong Completeness*: eventually *every* process that crashes is permanently suspected by *every* correct process.

Chandra and Toueg [CT96] proposed an algorithm to solve Consensus in an asynchronous model with crash-failures, where: (i) processes are fully interconnected through bidirectional *quasi-reliable point-to-point* communication channels, on which message delivery is only guaranteed if neither the sender nor the receiver crashes; (ii) a notion of *reliable broadcast* is built on top of quasi-reliable point-to-point channels to send messages to all processes in the system ensuring that all correct processes will eventually receive the messages. The algorithm generates a set of system runs constrained by:

- the hypothesis that only a minority of processes may crash
- the presence of the FD $\diamond\mathcal{S}$ to provide information about process crashes.

The algorithm is executed locally by each process and proceeds in *rounds*. It is based on the *rotating coordinator* paradigm: in each round r, a single process is predetermined to play a coordinator role while all other processes in the same round play the role of participants. Each of the n processes knows n and holds a local round counter; thus, at any time, it can locally determine the coordinator of any round. For every round, each participant sends its current value to the coordinator of this round. The coordinator chooses one of the most recent proposals among those that it has received and sends it to all participants. In turn, each participant is supposed to acknowledge the receipt updating its current value with the chosen value proposed by the coordinator. If the coordinator receives a majority of positive acknowledgments, it broadcasts the value to all participants using the Reliable Broadcast mechanism. This value is then delivered to all correct processes that can now decide and exit the algorithm. Since no assumption on time can be made, it is impossible for a participant to tell the reason (slow network or crashed coordinator) behind the non-reception of a message from the coordination. Thus, each participant has the possibility of suspecting a coordinator to have crashed. In this case, the participant sends a negative acknowledgment and moves to the following round, maintaining its current value.

A critique on pseudo code for verification purposes. Chandra and Toueg's [CT96] proofs of the Validity, Agreement and Termination properties are basically written in natural language.

Given that such proofs did not constitute the main result of their paper, one may find them reasonable. However, we found them nevertheless rather hard to follow and very hard to formally verify.

The first problem resides in the fact that the algorithm description is *not sufficiently complete*:

- it describes only local behaviors and does not capture the communication network behavior;
- the semantics of the pseudo code and its underlying services is at best specified informally;
- the involved data structures are under-specified.

The second problem is that the algorithm description *does not offer an unambiguous derivation of runs from the code*. This is crucial, because the specification of the correctness properties is exclusively based on the notion of system runs of the full distributed system, including quasi-reliable point-to-point and reliable broadcast message-passing, as well as the failure detection mechanism.

The third problem is more subtle, as it addresses the relation between the concepts of "time" and "round" in the context of process systems that execute round-based algorithms. Note that a round is only a local concept. There is nothing like *the* global round number of a reachable global state in a system run. Due to the asynchrony of the underlying model, any such process system may easily reach states in which all processes are in different rounds[1]. Now, the proofs of the Consensus properties—defined over runs—make heavy reference to the concept of rounds. However, the *relation between runs and asynchronous rounds is never properly clarified*. We consider this as problematic.

A typical proof technique employed by Chandra and Toueg in [CT96] is mathematical induction with respect to round numbers, while reasoning over run-based properties. Typically, such an induction starts with the smallest round in which some property X holds, e.g., in which a majority of processes has positively acknowledged. In a given run, to find this starting point one may take the initial state and search from there for the first moment in time in which X holds for some round. However, this procedure is not correct. It may well be that at a *later point in time* of the run, X holds for a *smaller round*. Accordingly, when the induction proceeds to a higher round, it might have to proceed "backwards in time" along a system run. Therefore, the concept of time—and of the respective iteration principles along the construction of a run—is not obviously compatible with the concept of asynchronous rounds. The solution, rather implicit in [CT96], is to ignore *when* exactly events happened, just noting *that* they happened. More precisely, one should pick a sufficiently advanced state of a given run (for example the last one in a finite run), and then find an appropriately abstract way to reason—by induction—about its *possible past*. Summing up, we believe that the proofs would profit much from a global view on system states and their past that provides us with precise information about which processes *have been* in which round in the past, and what they precisely did when they were there.

As an aside and general remark, we would like to point out that, also the implementers of distributed systems would greatly benefit from a formalization of the algorithms. Far too often the pseudo code description of an algorithm leaves unspecified the environmental behaviour (process failures, network properties, attached additional devices) under which a particular distributed algorithm actually works.

Contribution. We introduce some appropriate data structures to model quasi-reliable point-to-point communication, reliable broadcast, and process states. Taken together, these three data structures represent what we call the (global) *configuration of a system*. We then provide a formalization in our setting of Ω, a FD equivalent to $\diamond\mathcal{S}$ (as proved in [CHT96, NF03]) but more convenient for our purposes.

[1] Actually, the algorithm may easily reach system configurations in which, at a certain point in time, every process is coordinator in *its* current round, while all processes are in pairwise different rounds, by having every participant simply always suspect the respective coordinator. Analogously, the algorithm may easily reach moments in which none of the processes is the coordinator of its round. Moreover, in such a moment, it is impossible to predict, from a chronological point of view, which process will next become coordinator.

The dynamics of point-to-point communication, reliable broadcast, as well as process crashes, is modeled as a rule-based transition system. Modeling these concepts is mostly orthogonal to the Consensus algorithm which is also expressed in terms of transition rules. The proofs of the Validity, Agreement, and Termination properties rely on a number of crucial properties on systems runs and configurations. We often prove those properties by induction on either the round number or the length of (the derivation) of system runs; we pay particular attention to avoid mixing up the two proof techniques.

Outline. To prepare the ground, in §2, we informally introduce the distributed model underlying our study, including the specifications of all communication and coordination primitives that we require. Then, we present the chosen Consensus algorithm, both in pseudo code (in §3) and in terms of a rewrite system, formally defined by a set of transition rules (§4). In §5, we provide a number of crucial lemmas used in §6 to formally prove the Validity, Agreement, and Termination properties of Chandra and Toueg's algorithm. An evaluation of our work and comparison with other approaches is deferred to §7.

2 An asynchronous model of distributed processes

In the model underlying Chandra and Toueg's algorithm [CT96], a system consists of a set of n *processes* enumerated by $i \in \mathbb{P} = \{\, 1 \ldots, n \,\}$. Processes run independently of each other.

The behavior of a system is described as sequences of computation steps starting from some initial configuration. Individual steps correspond to actions of the communication network or to internal actions of the processes and there is a one to one correspondence between steps and actions. The step sequences are sometimes called *schedules* [CT96]. In this paper, we will refer to them as *runs*, as it is more common terminology in the context of operational semantics.

2.1 Quasi-reliable point-to-point communication

Processes are fully interconnected by bidirectional communication channels representing the underlying communication medium. The primitives in pseudo code are quite straightforward:

- "QPP_SEND (*sender*, *number*, *contents*) TO *receiver*" denotes quasi-reliable emission of a message from process *sender* to process *receiver*, where *number* plays the role of some sequence number to distinguish the messages coming from the sender;
- "QPP_RECEIVE(s, n, c)" serves as the reception counterpart extracting the relevant data into the indicated formal parameters s, n, c.

The property that is required on these operations is:

Quasi-reliability

If process *sender* executes "QPP_SEND (*sender*, *number*, *contents*) TO *receiver*", and both *sender* and *receiver* are correct, then *receiver* will eventually execute "QPP_RECEIVE(s, n, c)" to receive (*sender*, *number*, *contents*).

In their first version [CT96], Chandra and Toueg presented the channels as being *reliable*, which means that once a message is sent to a process that does not crash, it is guaranteed to be received no matter whether the sender crashes. However, in a later paper [ACT97], Chandra and Toueg referred to the channels used in [CT96] as being *quasi-reliable*, where message delivery is guaranteed only if also the sender stays alive. So, the difference between the reliable and quasi-reliable variants consists solely in the conditions imposed on the sender to guarantee the delivery of the messages (to non-crashed processes). In both cases, if a sender crashes, the delivery of previously sent messages is not forbidden, but may still be possible. More precisely, if a sender crashes after having performed a concurrent send action, then some messages belonging to that action may be successfully delivered, while others may be lost.

2.2 Failures and their unreliable detection

In the so-called *fail-stop* variant of the base model of §2.1, processes may crash, and when they do so, they do not recover from this state. A process that crashes in a run does no longer contribute to the system evolution in that run; it does neither send nor receive messages. As already said in the Introduction, a process is called *correct* (in a given run) if it does not crash (in this run). Note that processes need not to be consistently correct in all possible runs.

If distributed algorithms shall work in the context of possible process crashes, then the notion of *failure detection* or *failure suspicion* becomes useful. According to [CT96], a *failure detector* (FD) is an idealized abstraction that governs the allowed mutual suspicions of processes about whether other processes have crashed.[2]

The pseudo-code primitive to query a FD is the following:

- "suspected(q)". When executed by process p, with $p \neq q$ (a process cannot suspect itself), it returns a boolean value that tells whether the FD of process p currently suspects process q to have crashed.

Perfect (i.e., always reliable) failure detection is not implementable in purely asynchronous systems, since it is impossible to distinguish the processes that are crashed from those that are just slow. Chandra and Toueg [CT96] proposed to augment the asynchronous model of computation with *unreliable* failure detection, whose degree of reliability is imposed by means of explicit constraints.

A number of combinations of basic reliability constraints on FDs were proposed in [CT96]. The Consensus algorithm that we study in this paper was designed to work correctly when using the FD $\diamond\mathcal{S}$, which satisfies the following pair of constraints:

Eventual Weak Accuracy
There is a time after which *some* correct process is never suspected by *any* correct process.

Strong Completeness
Eventually *every* process that crashes is permanently suspected by *every* correct process.

It turns out that Chandra and Toueg's Consensus algorithm also works correctly when using another FD, denoted by Ω [CHT96]. This FD is characterized by a single property based on explicit *trust* rather than *suspicion*, which may roughly be seen as dual notions of information about the crash status of other processes.

Eventual Leadership
There is a time after which all the (correct) processes always trust the same correct process.

The FDs Ω and $\diamond\mathcal{S}$ are equivalent, in the sense that one can be used to implement the other, and vice versa [CHT96, NF03]. Thus, it is not that surprising that the Consensus algorithm also works with Ω. Although Ω was originally introduced only to simplify the proofs of [CHT96], it is more convenient to develop our formal model based on it rather than $\diamond\mathcal{S}$: instead of keeping track of big amounts of useless unreliable suspicion information, Ω records only small amounts of reliable trust information [NF03].

2.3 Reliable broadcast

Built on top of quasi-reliable point-to-point messaging, the notion of *reliable broadcast* will be conveniently used later on for the Consensus algorithm. Reliable broadcast allows to send a message to all the processes in the system, assuring that (at least) all the correct processes will eventually receive the message. The pseudo-code primitives are:

- "RBC_BROADCAST(*sender*, *contents*)" denotes the broadcasting of a message to all processes, so no receiver indication is required;
- "RBC_DELIVER(s, c)" serves as the reception counterpart extracting the relevant data into the indicated formal parameters s, c.

[2] Less abstractly, one would provide n local FD devices that are attached to the n individual processes.

Reliable broadcast is characterized by the following three properties:

Validity
: If a correct process executes RBC_BROADCAST(*sender*, *contents*), then it also eventually executes RBC_DELIVER(s, c) for the message (*sender*, *contents*).

Agreement
: If a correct process executes RBC_DELIVER(s, c) for some message (*sender*, *contents*), then all correct processes eventually execute RBC_DELIVER(s, c) for this message (*sender*, *contents*).

Uniform Integrity
: For any message (*sender*, *contents*), every process executes RBC_DELIVER(s, c) at most once, and only if some process previously executed RBC_BROADCAST(*sender*, *contents*).

2.4 The Consensus specification

The essential idea of Distributed Consensus is that initially each process proposes some value v, and eventually all non-crashed processes agree on one of the proposed values. The primitives are:

- "C_PROPOSE(v)", and
- "C_DECIDE(v)".

In summary, we recall that each run of Consensus must satisfy the following three properties:

Validity
: If a process decides some value v (i.e., executes C_DECIDE(v)), then some process must have proposed that value (i.e., executed C_PROPOSE(v)).

Agreement
: No two correct processes decide (i.e., execute C_DECIDE for) different values.

Termination
: Every correct process (eventually) decides (i.e., executes C_DECIDE for) some value[3].

3 The Consensus algorithm by Chandra and Toueg

In this section, we explain the details of the Consensus algorithm proposed by Chandra and Toueg in [CT96]. The correctness of the algorithm depends on the availability and correct functioning of all the lower-level services presented in the previous section: quasi-reliable point-to-point communication, reliable broadcast and the FD $\diamond\mathcal{S}$.

The algorithm proceeds in *asynchronous rounds*: each of the n processes counts rounds locally and advances independently in the execution of the algorithm. Furthermore, the algorithm is based on the *rotating coordinator* paradigm: for each round number r, a single process is predetermined to play a coordinator role, while all other processes in the same round play the role of *participants*. Let n be the number of processes, then the function crd(r) returns the integer $((r-1) \bmod n)+1$ denoting the coordinator of round r. This function is known to all the processes, therefore each of them is able to compute at any time the identity of the coordinator of a round.

Each of the n processes stores a local *estimate* of the sought decision value, together with a *stamp* that tells when—i.e., in which round—the process has come to believe in this estimate. Let us refer to the pair (*estimate*,*stamp*) as a *belief*. The goal of the algorithm is to reach a situation in which a majority of processes agree on the same estimate. To reach such a situation, coordinators and participants exchange their current beliefs, round after round, through QPP-messages. The role of a coordinator is (i) to collect sufficiently many current beliefs, (ii) to find out the most recent estimates by comparing the respective round stamps, (iii) to select one estimate among those, and (iv) to disseminate its selection to the participants such that they can update their local beliefs accordingly.

[3] As noted by Chandra and Toueg [CT96], the algorithm under consideration satisfies also a property called *Uniform Agreement* that strengthens the property of Agreement to hold for *all* the processes in the system, independently of whether they are correct or not.

3.1 The algorithm in pseudo-code format

In this paper, we provide a Consensus algorithm in pseudo code, that is slightly different from the one used by Chandra and Toueg, but that is more suitable to be formalized as we will do later on. The differences between the two versions of pseudo code is mainly due to the data structures involved. A more detailed analysis is given at the end of this section.

Keywords. We use the programming constructs `while` and `if·then·else·` with their standard interpretation. We also use the construct `await` to indicate that execution is suspended until some associated boolean condition is satisfied, and we use `when` to indicate that the execution of some associated primitive triggers the execution of the subsequent code.

Data structures. Each process carries a state containing the following three ingredients:

- 'counter': storing the local round number,
- 'belief': storing the pair (or binary record) of 'estimate' and 'stamp', and
- 'decision': to control the exit condition of a `while`-loop.

We consider process states as arrays of records. When we are interested in selective extraction and update, we treat a process state explicitly as a record with the labels indicated above. So, we use x.label to denote record selection; we use $x \leftarrow v$ to denote assignment of all the record fields of x, and x.label $\leftarrow v$ to denote assignment of label of x.

Program structure. The pseudo code of Table 1 describes the behavior of any process i, for $i \in \mathbb{P}$. Any process i locally starts a Consensus run by executing C_PROPOSE(v_i), where v_i is taken from a set $\mathbb{V}$ of proposable values. The process first initializes its local state: the 'counter' and the 'stamp' are initialized to zero, the 'estimate' is initialized to v_i, the 'decision' is left undefined, written $\perp$. Then the process enters a `while`-loop. When starting a Consensus run, the process launches also a concurrent sub-routine that can modify the value of the state component 'decision', execute C_DECIDE and prevent the `while`-loop from restarting.

Each cycle through the `while`-loop represents a round. It proceeds in steps (labeled P1...P4) that represent the four phases of a round. Before explaining in detail the structure of the `while`-loop, let us have a look at QPP-messaging and Reliable Broadcast.

QPP-messaging. In QPP_SEND (*sender*, (*round*, *phase*), *contents*) TO *receiver* the pair (*round*, *phase*) plays the role of a sequence number, which, together with the *sender* and *receiver* identification, allows us to distinguish—regardless of the *contents*—all QPP-messages ever sent during a Consensus run, under the simple condition that every process sends to any other process at most one message per (*round*, *phase*). The type of *contents* depends on the respective *phase*: it may be a belief (phase P1 or P2) or an `ack`/`nack` (phase P3). Note that the algorithm never sends QPP-messages in phase P4.

A QPP-message travels through the network from the sender to the receiver. By the assumption of quasi-reliability, the reception is guaranteed if neither the sender nor the receiver crash. More precisely, if the sender crashes after sending, then the message may still be received at some point, but there is simply no guarantee for this event to ever happen. One might expect the pseudo code to feature explicit executions of QPP_RECEIVE to describe individual receptions. However, since our pseudo code often blocks until certain messages have arrived, and since it sometimes acts if a particular message has not yet arrived, it makes sense to silently add incoming messages, on reception, to some local *message buffer* (implicitly defined). In order to express the fact that the algorithm waits for a particular condition on the message buffer to become true, our pseudo code features the use of an `await` construct. As appropriate buffering data structures we introduce sets denoted by received_QPP, for which we conveniently also provide the notation received_QPP.P^r that extracts the received messages, according to *round* $r \in \mathbb{N}$ and *phase* $P \in \{\mathsf{P1}, \mathsf{P2}, \mathsf{P3}\}$. Note

```
PROCEDURE C_PROPOSE (v_i)
  state.counter ← 0
  state.belief.value ← v_i
  state.belief.stamp ← 0
  state.decision ← ⊥

  while state.decision = ⊥
  {  state.counter ← state.counter + 1
     r ← state.counter

     P1   QPP_SEND (i, (r, P1), state.belief) TO crd(r)

     P2   if      i = crd(r)
          then    { await   | received_QPP.P1^r | ≥ ⌈(n+1)/2⌉
                    for 1 ≤ j ≤ n
                    do  QPP_SEND (i, (r, P2), best(received_QPP.P1^r)) TO j
                  }

     P3   await  (  received_QPP.P2^r ≠ ∅  or  suspected(crd(r))  )
          if      received_QPP.P2^r = {(crd(r), (r, P2), (v, s))}
          then    { QPP_SEND (i, (r, P3), ack) TO crd(r)
                    state.belief ← (v, r)
                  }
          else      QPP_SEND (i, (r, P3), nack) TO crd(r)

     P4   if      i = crd(r)
          then    { await   | received_QPP.P3^r | ≥ ⌈(n+1)/2⌉
                    if      | ack(received_QPP.P3^r) | ≥ ⌈(n+1)/2⌉
                    then    RBC_BROADCAST(i, state.belief)
                  }
  }


when  RBC_DELIVER(j, d)
{  if      state.decision = ⊥
   then    state.decision ← d
           C_DECIDE (state.decision.value)
}
```

Table 1: Consensus via Pseudo Code

that each process i maintains its own collection of such sets, and that no process j can access the collection of another process i.

Reliable broadcast. In contrast to QPP-messaging, where we actively await the arrival of suitable messages of certain kinds, no receiver-side buffering is required for the treatment of RBC_DELIVER. Thus, the reception of broadcast messages is performed individually and directly using `when`-clauses.

Loop structure. As said before, every process goes through four phases per round, i.e., per `while`-cycle. In the following, when we speak of a process executing an action, we implicitly mean a correct process.

In phase P1, each process sends to the coordinator its current belief, i.e., a pair consisting of an estimate and its stamp.

In phase P2, all the processes except for the coordinator move immediately to phase P3. The coordinator waits until it receives sufficiently many proposals (here, we use the standard notation $|\cdot|$ to denote the cardinality of sets), namely at least $\lceil (n+1)/2 \rceil$, and selects among them the one with the highest 'stamp'. Here, best(received_QPP.P1^r) indicates a (local) function that performs this operation. This "best" proposal is called the *round proposal* and is distributed to all participants, again using QPP-messages. The coordinator then also moves to phase P3.

In phase P3, each process waits for a moment in which either the coordinator's round proposal has arrived at the local site or the coordinator is suspected by the process's (local) FD. In both cases, the process then sends an acknowledgment (positive: `ack` or negative: `nack` respectively) to the coordinator and proceeds to the next phase. Only in the positive case, the process adopts the round proposal. Recall that, by our definition of FD, a process (and in particular the coordinator) cannot suspect itself. As a consequence a coordinator will always adopt the round proposal and send a positive acknowledgment to itself.

In phase P4, all the processes except for the coordinator proceed to the next round by `while`-looping. The coordinator waits until it receives sufficiently many acknowledgments. Here, ack(received_QPP.P3^r) selects the subset of received_QPP.P3^r that contains all the received messages carrying a positive acknowledgment. If the cardinality of ack(received_QPP.P3^r) is at least equal to $\lceil (n+1)/2 \rceil$ then we say that the value proposed by this majority is *locked* and the coordinator "initiates a decision", otherwise the coordinator fails this occasion and simply ends this round. In both cases, the coordinator then proceeds to the next round by re-entering the `while`-loop.

The above-mentioned "initiating a decision" means to execute RBC_BROADCAST(i, *state*.belief), by which the round proposal that has just been adopted by a majority of processes, is transmitted to *all* processes. The corresponding receptions RBC_DELIVER(j, d) are captured by the `when`-clause that is running concurrently: processes may RBC_DELIVER regardless of their current round and phase. On execution of RBC_DELIVER(j, d), and only when it happens for the first time, the process "officially" announces its decision for the delivered value by executing C_DECIDE. Although it is not explicitly indicated by Chandra and Toueg, the `when`-clause must be performed atomically to guarantee Termination. Notice also that, by the usual convention on Reliable Broadcast, also the broadcast initiator must itself explicitly perform the delivery before it can decide. Since Reliable Broadcast satisfies the Agreement property, every non-crashed process will eventually receive the broadcast message.

Differences to the version of Chandra and Toueg. Apart from slightly different keywords, our pseudo code deviates from the original one in the following minor ways. (1) We chose to compact all variables that contribute to the state of a process into a single data structure. (2) We chose a more abstract primitive to indicate process suspicion, while the original code directly referred to the author's mathematical model of FD. (3) We use the FD Ω instead of the original $\diamond\mathcal{S}$. (4) We use explicit proper data structures (instead of leaving them implicit) to deal with

received QPP-messages, to select the subsets ack(received_QPP.P^r) for $P \in \{\mathsf{P1}, \mathsf{P3}\}$, and we also introduce the operation best(received_QPP.$\mathsf{P1}^r$) to operate on these structures.

3.2 An intuitive argument of correctness

As argued in the Introduction, the algorithm works correctly satisfying Validity, Agreement, and Termination.

Validity holds because none of the clauses of the algorithm invents any value. Whenever a value is written, it is always *copied* from some other data structure, or selected among several such sources. Obviously then, all values that occur in a reachable system configuration must originate from some initially given value.

According to Chandra and Toueg, Agreement and Termination can intuitively be explained by means of a three-epoch structure of runs. In epoch 1 everything is possible, i.e., every initial value might eventually be decided. Moving to epoch 2 means that a value *gets locked*, i.e., for some round r (actually: the smallest such) a majority of processes send a positive acknowledgment to the coordinator of r. Since the coordinator uses a maximal time-stamp strategy when choosing its proposal, and since only a minority of processes may crash, a locked value v will always dominate the other values: should some value get locked in a greater round, then it will be the very same value v that was previously locked. In epoch 3 all (correct) processes decide the locked value. Due to the Reliable Broadcast properties, if a correct process decides, then all other correct processes decide as well.

Agreement is a safety property which guarantees that nothing goes wrong. In this case it holds because (1) if two values v_1 and v_2 are locked in two different rounds, then $v_1 = v_2$; (2) if a process decides, it always decides on a locked value. Termination is a liveness property which guarantees that something good eventually happens. In this case it holds since the FD properties enforce the eventual transition of the system from epoch 1 to epoch 2, and from epoch 2 to epoch 3.

The proof of Termination relies on the combination of two sub-results (1) *correct processes do not get blocked forever*, and (2) *some correct process eventually performs some crucial action.* As to (1), the requirement that only a minority of processes can crash ensures that *coordinators do not block forever* when waiting for a majority of messages (in phases P2 and P4); moreover, the Strong Completeness property of $\diamond\mathcal{S}$ ensures that all the crashed processes will be eventually suspected; in particular *no process can block forever waiting for a crashed coordinator.* As to (2), the Eventual Weak Accuracy property ensures that eventually *some correct coordinator will be no longer suspected*, will receive a majority of positive acknowledgments, and will RBC-broadcast the best value; the RBC-properties then ensure that all values that are RBC-broadcast by coordinators will eventually be RBC-delivered (and hence decided) by all correct processes.

4 Formalizing the algorithm

In the Introduction we discussed the lack of formality in the description of the algorithm when using pseudo code. In order to counter this incompleteness, we need to develop appropriate description techniques that incorporate the lacking information. To counter the observed ambiguity in the semantics of the pseudo code with respect to the underlying system model, we propose to build the algorithm upon a formal description.

The approach that we pursue is to provide a global syntax-free description of the algorithm that is directly built upon the mathematical definition of an enhanced underlying system model. Thereby, the gap between any kind of syntax and its run-time operational semantics is conveniently avoided such that we can focus on the actual proofs on the mathematical objects themselves. Technically, we are going to use transition rules, resulting in a description that resembles rewrite systems.

In order to provide a complete account of the behavior of the Consensus algorithm, we need to explicitly represent the communication network as well as all interactions with it, in particular the buffering of QPP- and RBC-messages. Thus, in essence, we provide a mathematical structure

that describes the global idealized run-time system comprising all processes and the network. In order to simplify the proofs, our structure will also play the role of a system history that keeps track of relevant information during computation.

Formally, the consensus algorithm is defined in terms of transition rules, that define computation steps as transitions between configurations of the form:

$$\langle\, \mathbf{B}, \mathbf{Q}, \mathbf{S} \,\rangle \;\rightarrow \langle\, \mathbf{B}', \mathbf{Q}', \mathbf{S}' \,\rangle \qquad \text{or, in vertical notation:} \qquad \left\langle \begin{array}{c} \mathbf{B} \\ \mathbf{Q} \\ \mathbf{S} \end{array} \right\rangle \rightarrow \left\langle \begin{array}{c} \mathbf{B}' \\ \mathbf{Q}' \\ \mathbf{S}' \end{array} \right\rangle$$

where $\mathbf{B}$ and $\mathbf{Q}$ contain the (histories of) messages sent during the execution of the algorithm through the Reliable *B*roadcast service and through the *Q*uasi-reliable Point-to-Point service, respectively, while $\mathbf{S}$ models the *state* of the n processes.

The symbol $\rightarrow^*$ denotes the reflexive-transitive closure of $\rightarrow$ (formally defined in Tables 2 and 3). The symbol $\rightarrow^\infty$ denotes an infinite sequence of $\rightarrow$. A *run* of the Consensus algorithm is then represented by sequences

$$\langle\, \mathbf{B}_0, \mathbf{Q}_0, \mathbf{S}_0^{(v_1 \ldots, v_n)} \,\rangle \;\rightarrow^* \langle\, \mathbf{B}, \mathbf{Q}, \mathbf{S} \,\rangle \;\rightarrow^\imath \qquad \text{where } \imath \in \{\, *, \infty \,\}$$

where $\langle\, \mathbf{B}_0, \mathbf{Q}_0, \mathbf{S}_0^{(v_1, \ldots, v_n)} \,\rangle$ denotes the initial configuration based on the values $v_1, \ldots, v_n$ initially proposed by the n processes in the system. We require runs to be complete, i.e., either infinite, or finite but such that they cannot be extended by some computation step in their final configuration.

4.1 Configurations

Next, we explain in detail the three components of a *configuration* $\langle\, \mathbf{B}, \mathbf{Q}, \mathbf{S} \,\rangle$.

Process State

When representing the algorithm by a set of step-oriented transition rules, we inevitably lose the explicit sequencing of statements in the former pseudo-code description of the algorithm. To remedy this loss of information, we model explicitly to which subsequent phase the algorithm needs to jump after the application of any transition rule. Thus, we enhance the "program counter" of individual processes to include not only their round number r, but also the label indicating their current phase P. We further introduce an additional label W, which we use to model the "entry phase" of `while`-loops.

Component $\mathbf{S}$ is defined as an array of n elements, where $\mathbf{S}(i)$, for $1 \leq i \leq n$, represents the state of process i. The state of a process is a triple of the form $\big(c,\, b,\, d\big)$, where

- $c = (r, P)$ is the *program counter* (always defined), consisting of a round number $r \in \mathbb{N}$ and a phase label $P \in \{\mathsf{W}, \mathsf{P1}, \mathsf{P2}, \mathsf{P3}, \mathsf{P4}\}$;
- $b = (v, s)$ is the *belief* (always defined), consisting of a value estimate $v \in \mathbb{V}$ and a stamp $s \in \mathbb{N}$, representing the round in which the belief in the value was adopted;
- $d = (v, s)$ is the *decision* (may be undefined, i.e., $d = \bot$), consisting of a value $v \in \mathbb{V}$ and a stamp $s \in \mathbb{N}$, representing the round in which value v has been broadcasted by $\mathrm{crd}(s)$.

Every process i starts at round 0 and in phase W, where it simply believes in its own value v_i tagged with stamp 0; the decision field is left undefined. Thus, for every process i, the initial state is:

$$\mathbf{S}_0^{(v_1, \ldots, v_n)}(i) = \big((0, \mathsf{W}), (v_i, 0), \bot\big)$$

We write $\mathbf{S}\{\, i \mapsto V \,\}$ for $i \in \mathbb{P}$ and $V \in ((\mathbb{N} \times \{\mathsf{W}, \mathsf{P1}, \mathsf{P2}, \mathsf{P3}, \mathsf{P4}\}) \times (\mathbb{V} \times \mathbb{N}) \times ((\mathbb{V} \times \mathbb{N}) \cup \{\bot\})) \cup \{\bot\}$ to insert/update entries in $\mathbf{S}$ as follows:

$$\mathbf{S}\{\, i \mapsto V \,\}(k) \stackrel{\text{def}}{=} \begin{cases} V & \text{if } k = i \\ \mathbf{S}(k) & \text{otherwise} \end{cases}$$

$$(\textsc{crash})\ \frac{\mathbf{S}(i) \neq \bot}{\langle\, \mathbf{B}, \mathbf{Q}, \mathbf{S} \,\rangle\ \rightarrow \langle\, \mathbf{B}, \mathbf{Q}, \mathbf{S}\{\, i \mapsto \bot \,\} \,\rangle}$$

$$(\textsc{Qpp\_snd})\ \frac{\mathbf{S}(j) \neq \bot \qquad X \in \{\, \mathsf{P1}^r_{j\to k}, \mathsf{P2}^r_{j\to k}, \mathsf{P3}^r_{j\to k} \,\} \qquad \mathbf{Q}.X = (w, \text{outgoing})}{\langle\, \mathbf{B}, \mathbf{Q}, \mathbf{S} \,\rangle\ \rightarrow \langle\, \mathbf{B}, \mathbf{Q}\{\, X \mapsto (w, \text{transit}) \,\}, \mathbf{S} \,\rangle}$$

$$(\textsc{Qpp\_deliver})\ \frac{\mathbf{S}(k) \neq \bot \qquad X \in \{\, \mathsf{P1}^r_{j\to k}, \mathsf{P2}^r_{j\to k}, \mathsf{P3}^r_{j\to k} \,\} \qquad \mathbf{Q}.X = (w, \text{transit})}{\langle\, \mathbf{B}, \mathbf{Q}, \mathbf{S} \,\rangle\ \rightarrow \langle\, \mathbf{B}, \mathbf{Q}\{\, X \mapsto (w, \text{received}) \,\}, \mathbf{S} \,\rangle}$$

Table 2: Qpp-Message Transmission and Process Crash

In Table 2, with rule (Crash), we model the crash of a process i by setting the respective ith entry of $\mathbf{S}$ to $\bot$. Note that the rule is defined such that any non-crashed process can crash at any time. The only premise of the rule is the state of the process being defined, regardless of the precise value. Note also that each execution of (Crash) allows only one process to crash.

Qpp-messaging

The component $\mathbf{Q}$ is a *global* data structure that contains Qpp-messages and their transmission status. The idea is that a particular $\mathbf{Q}$ is associated to a particular instant in time referring to a particular run. More precisely, an instantaneous $\mathbf{Q}$ contains all Qpp-messages that have been sent (and possibly received) by the n processes up to that instant. This representation will be convenient for both the formal description of the algorithm and its formal verification.

In order to keep track of all Qpp-messages ever sent during a run of the Consensus algorithm, we might use a (multi-)set that is suitably defined as follows: for every execution of an instruction

$$\textsc{Qpp\_send}\ (j, (r, P), \mathit{contents})\ \textsc{to}\ k$$

we might record the information as a tuple of the form

$$(r, P, j, k, \mathit{contents})\ .$$

However, this is not enough to model the lifetime of Qpp-messages, which pass through various transmission states that need to be distinguished. According to Chandra and Toueg's model, if a sender crashes after having performed a concurrent send action, then some messages belonging to that action may be successfully delivered, while others may be lost. It turns out that we can easily accomplish this goal by the use of the following three transmission states:

outgoing**:** messages that, by the execution of some Qpp_send statement, are put into an unordered buffer for *outgoing* messages at the sender site; the intuition is that such messages have not yet left the sender site, so a sender crash might still make them unavailable forever.

transit**:** messages that have left the sender site, but that have not yet arrived at their destination; these messages are kept in the unordered buffer of the transmission medium; the intuition is that such messages have already left the sender site and are available in the network, so they can be received even in case of a sender crash;

received**:** messages that—corresponding to the execution of some Qpp_receive statement—are stored within an unordered buffer for *received* messages at the intended receiver site.

In our data structure $\mathbf{Q}$, we model these transmission states via the respective tags outgoing, transit, and received. In turn, this requires us to use (instead of the version above) extended tuples of the form

$$(r, P, j, k, \mathit{contents}, \tau)$$

where $\tau \in \{\, \text{outgoing}, \text{transit}, \text{received} \,\}$. Note that, in our Consensus system, component $\mathbf{Q}$ is not associated to a single location since it contains QPP-messages in all of its transmission states:

- by including message data of the form $(\cdot, \text{transit})$ it captures the *non-local* state of the point-to-point medium that is seen as a buffer for the messages in transit from one site to another;
- by including message data of the form $(\cdot, \text{outgoing})$ and $(\cdot, \text{received})$, it represents the *local* unordered buffers of outgoing/received messages at the sender/receiver site.

According to the usage patterns of QPP-messages in the Consensus algorithm, we may observe that in any round r and phase P, there is at most one message ever sent from process j to process k (see Lemma 5.2.3 of message uniqueness). Thus, it suffices to model the occurring QPP-messages with ordinary sets instead of multisets. Taking further advantage of this uniqueness property to conveniently manipulate the data structure $\mathbf{Q}$, we introduce the notation of "structured" *selectors* (representing the message's control information) of the form

$$P^r_{j \to k}$$

for $P \in \{\, \mathsf{P1}, \mathsf{P2}, \mathsf{P3} \,\}$, $r \in \mathbb{N}$, and $j, k \in \mathbb{P}$. Recall from the pseudo code that processes never send QPP-messages in phase $\mathsf{P4}$ (Table 1), so we do not need an entry for such phase. Using structured selectors, we extract data from entries in $\mathbf{Q}$ using the notation:

$$\mathbf{Q}.P^r_{j \to k} \stackrel{\text{def}}{=} \begin{cases} (\mathit{contents}, \tau) & \text{if } (r, P, j, k, \mathit{contents}, \tau) \in \mathbf{Q} \\ \bot & \text{otherwise} \end{cases}$$

We insert/update entries in $\mathbf{Q}$ by the notation $\mathbf{Q}\{\, X \mapsto V \,\}$ for $X \in \{\, \mathsf{P1}^r_{j \to k}, \mathsf{P2}^r_{j \to k}, \mathsf{P3}^r_{j \to k} \,\}$ and $V \in (((\mathbb{V} \times \mathbb{N}) \cup \{\text{ack}, \text{nack}\}) \times \{\text{outgoing}, \text{transit}, \text{received}\})$ defined by:

$$\mathbf{Q}\{\, X \mapsto V \,\}.P^r_{j \to k} \stackrel{\text{def}}{=} \begin{cases} V & \text{if } X = P^r_{j \to k} \\ \mathbf{Q}.P^r_{j \to k} & \text{otherwise} \end{cases}$$

Using this notation, the rules (QPP_SND) and (QPP_DELIVER) in Table 2 model the lifetime of messages across transmission states. Note that these rules are independent of the current components $\mathbf{B}$ and $\mathbf{S}$ (in fact, $\mathbf{S}$ is only used to ensure that the process involved in the sending has not crashed). This reflects the fact that the underlying transmission medium ships messages from node to node, independently of any algorithm running at the nodes of the system. Note that there is no correspondence between these rules and the pseudo code of §3.1, because the latter simply ignores such details.

Further analyzing the usage patterns of QPP-messages within the Consensus algorithm, we observe that selectors $P^r_{j \to k}$ are always of the form $\mathsf{P1}^r_{i \to \text{crd}(r)}$, $\mathsf{P2}^r_{\text{crd}(r) \to i}$, or $\mathsf{P3}^r_{i \to \text{crd}(r)}$. Note the different role played by i in the various cases (sender in $\mathsf{P1}$ and $\mathsf{P3}$, receiver in $\mathsf{P2}$). Note also that one of the abstract place-holders j, k always denotes the coordinator $\text{crd}(r)$ of the round r in which the message is sent. Thus, one among the abstract place-holders j, k is actually redundant—which one depends on the phase of the message—but we stick to the verbose notation $P^r_{j \to k}$ for the sake of clarity.

For the data part, we mostly use the array formulation of *contents*.

- $\mathbf{Q}.\mathsf{P1}^r_{i \to \text{crd}(r)}$ has the form $((v, s), \tau)$, where (v, s) is the *proposal* of process i at round r;
- $\mathbf{Q}.\mathsf{P2}^r_{\text{crd}(r) \to i}$ has the form $((v, s), \tau)$, where (v, s) is the *round proposal* sent by coordinator $\text{crd}(r)$ to all processes $i \in \mathbb{P}$;
- $\mathbf{Q}.\mathsf{P3}^r_{i \to \text{crd}(r)}$ has the form (a, τ) where $a \in \{\, \text{ack}, \text{nack} \,\}$ is the positive/negative acknowledgment of process i at round r.

Initially, $\mathbf{Q}_0 = \emptyset$, because we assume that no message has been sent yet.

Reliable Broadcast

Reliable Broadcast is an abstraction that allows a process to reliably send a message to all the other (non-crashed) processes in the system, thus satisfying the properties of Validity, Agreement, and Integrity as reported in section 2.3. In order to satisfy the RBC-properties our model must keep track of the delivery status of each broadcast message with respect to every involved process. Thus, in a first approximation, we can see the component $\mathbf{B}$ as a (multi-)set in which each element contains (i) the broadcast message and (ii) the set of processes that are supposed to deliver it.

Note that the Consensus algorithm uses Reliable Broadcast exclusively for the dissemination of decisions: in a given round, only the coordinator can execute RBC_BROADCAST to trigger the sending of messages that mention this round number. Therefore, we use round numbers to properly distinguish all the broadcasts that may possibly occur in a single run. We model $\mathbf{B}$ as an unbounded array indexed by the round number $r \in \mathbb{N}$: $\mathbf{B}(r)$ is either undefined ($\perp$) or a pair of the form (v, D) where $v \in \mathbb{V}$ is the decision value broadcasted by $\mathrm{crd}(r)$, and $D \subseteq \mathbb{P}$ the set of processes that have not yet executed RBC_DELIVER for this message. Initially $\mathbf{B}_0 := \emptyset$, that is $\mathbf{B}_0(r) = \perp$ for all $r \in \mathbb{N}$. We insert/update entries in $\mathbf{B}$ by the notation $\mathbf{B}\{\, r \mapsto V \,\}$ for $r \in \mathbb{N}$ and $V \in (\mathbb{V} \times 2^{\mathbb{P}})$ defined by:

$$\mathbf{B}\{\, r \mapsto V \,\}(\hat{r}) \stackrel{\text{def}}{=} \begin{cases} V & \text{if } \hat{r} = r \\ \mathbf{B}(\hat{r}) & \text{otherwise} \end{cases}$$

4.2 Auxiliary notation

We often use a dot $\cdot$ as a wildcard in various expressions to indicate that we assume there exists an appropriate instance, but that its actual value does not matter in that context.

In the following, we define precise projections from $\mathbf{Q}$ onto selected sub-structures. Let $Q \subseteq \mathbf{Q}$, then

$$Q.P^r \stackrel{\text{def}}{=} \{\, x \in Q \mid x = (r, P, \cdot, \cdot, \cdot, \cdot) \,\}$$

is used to project out of a given structure Q the slice that describes just those elements that were exchanged in phase P of round r between arbitrary processes. Apart from this abbreviation, which works solely on the selector part of $\mathbf{Q}$'s elements, we also use some specific projections with respect to the data part.

$$\begin{aligned} \mathrm{received}(Q) &\stackrel{\text{def}}{=} \{\, x \in Q \mid x = (\cdot, \cdot, \cdot, \cdot, \cdot, \mathrm{received}) \,\} \\ \mathrm{ack}(Q) &\stackrel{\text{def}}{=} \{\, x \in Q \mid x = (\cdot, \cdot, \cdot, \cdot, \mathrm{ack}, \cdot) \,\} \end{aligned}$$

Note that all of these resulting slices are just subsets of Q, so it makes sense to permit the selector notation $Q.P^r_{j \to k}$ also for such slices of Q. These abbreviations will be used later on in the formalization of the Consensus algorithm to check their relation to the majority of processes.

On a lower level, we abbreviate the extraction of information out of 1st- and 2nd-phase message contents as follows: we use $\mathrm{prop}(((v, s), t)) := (v, s)$ to extract the proposal of a history element, while $\mathrm{stamp}(((v, s), t)) := s$ denotes just the stamp associated to the proposal.

In phase P2, the coordinator of a round selects the best proposal among a majority of 1st-phase proposals for that round. The best proposal is the one with the highest stamp. If there exist more than one proposal with the same stamp, different strategies can apply. In this paper, we select the proposal sent by the process with the smallest identifier. Other strategies can be reduced to this particular one by appropriately reordering processes.

Definition 4.2.1 *Let* $\langle\, \mathbf{B}, \mathbf{Q}, \mathbf{S} \,\rangle$ *be a configuration. Let* min *and* max *denote the standard operators to calculate the minimum and maximum, respectively, of a set of integers. Let*

$$\mathrm{highest}^r(\mathbf{Q}) \stackrel{\text{def}}{=} \max\{\ s \mid (\cdot, \cdot, \cdot, \cdot, (\cdot, s), \cdot) \in \mathrm{received}(\mathbf{Q}).\mathsf{P1}^r\ \}$$

denote the greatest (i.e., most recent) stamp occurring in a proposal that was received in round r. Let

$$\text{winner}^r(\mathbf{Q}) \stackrel{\text{def}}{=} \min\{\ j \mid (\cdot,\cdot,j,\cdot,(\cdot,\text{highest}^r(\mathbf{Q})),\cdot) \in \text{received}(\mathbf{Q}).\mathsf{P1}^r\ \}$$

*denote the process that contributes the winning proposal for round r in $\mathbf{Q}$.*P1*. Then,*

$$\text{best}^r(\mathbf{Q}) \stackrel{\text{def}}{=} \text{prop}(\mathbf{Q}.\mathsf{P1}^r_{\text{winner}^r(\mathbf{Q})\to\text{crd}(r)})$$

denotes the respective "best proposal" for round r, meaning the received proposal with the highest stamp sent by the process with the smallest identifier.

In phase P2, the coordinator process $\text{crd}(r)$ also sends $\text{best}^r(\mathbf{Q})$ as the round proposal for round r to all processes. When this happens, we denote the value component of the round proposal as follows.

Definition 4.2.2

$$\text{val}^r(\mathbf{Q}) \stackrel{\text{def}}{=} \begin{cases} v & \textit{if, for all } i \in \mathbb{P},\ \mathbf{Q}.\mathsf{P2}^r_{\text{crd}(r)\to i} = ((v,\cdot),\cdot) \\ \bot & \textit{otherwise} \end{cases}$$

Note that $\text{val}^r(\mathbf{Q})$ is always uniquely defined, thus it can also be seen as a function.

4.3 The algorithm in transition-rule format

Everything is in place to reformulate the Consensus algorithm in terms of transition rules. Recall from § 4.1 that the global initial state of the formalized Consensus algorithm is:

$$\langle\, \mathbf{B}_0, \mathbf{Q}_0, \mathbf{S}_0^{(v_1,\ldots,v_n)} \,\rangle \stackrel{\text{def}}{=} \langle\ \emptyset, \emptyset, \{\ i \mapsto ((0,\mathsf{W}),(v_i,0),\bot)\ \}_{i\in\mathbb{P}}\ \rangle\ .$$

The rules are given in Table 3. Intuitively, rule (WHILE) models the `while` condition of the algorithm; the rules (PHS-*) describe how processes evolve, *sequentially*, from one phase to the next, and from one round to the next; the rule (RBC-DELIVER) models the possibility of RBC-delivering a message that has previously been RBC-broadcast by some coordinator. Our verbose explanations below are intended to show in detail how closely these rules correspond to the pseudo code of §3.1. Finally, note that the rules of Table 3 are to be used together with the algorithm-independent rules of Table 2 on point-to-point message transmission and process crashes.

Every rule typically picks one of the processes, say i, and, driven by i's current program counter, checks whether the respective "firing conditions" are satisfied. By definition, the rules can only be applied to non-crashed processes i, i.e., when $\mathbf{S}(i) \neq \bot$.

A process can execute rule (WHILE) only while being in phase W and having not yet decided. A process is in phase W when the algorithm is first started and also, as we will see by examining the other rules, when a round finishes and a new one is ready to begin. The execution of rule (WHILE) affects only the state of the process, in particular its program counter: the round number is incremented "by one" and the phase becomes P1. A new round starts. This is the only rule that can be executed by a process that begins the algorithm.

Rule (PHS-1) implements phase P1 of Table 1. The process can execute this rule only when being in phase P1. The execution has the effect of creating an outgoing 1st-phase message directed to the coordinator of the round, and of incrementing the state of the process to a new phase P. The value of P depends on whether the process is the coordinator of the round or not, and it is equal respectively to P2 or P3. To optimize the presentation of the semantics, rule (PHS-1) is not exactly the representation of P1 as it already performs the coordinator check that in the pseudo code is only made in P2.

Rule (PHS-2) represents phase P2 of the algorithm in Table 1. As said above, the rule does not check for coordinator identity because this test is already performed in rule (PHS-1). Consequently, only the coordinator has the possibility of being in phase P2 of a round. In order to execute this rule, the process must be in phase P2 and must have received at least a majority of the 1st-phase

$$\text{(WHILE)}\ \frac{\mathbf{S}(i) = ((r, \mathsf{W}), b, \perp)}{\left\langle \begin{array}{l}\mathbf{B}\\ \mathbf{Q}\\ \mathbf{S}\end{array} \right\rangle \rightarrow \left\langle \begin{array}{l}\mathbf{B}\\ \mathbf{Q}\\ \mathbf{S}\{\ \ i \ \mapsto\ ((r{+}1, \mathsf{P1}), b, \perp)\ \ \}\end{array} \right\rangle}$$

$$\text{(PHS-1)}\ \frac{\mathbf{S}(i) = ((r, \mathsf{P1}), b, d)}{\left\langle \begin{array}{l}\mathbf{B}\\ \mathbf{Q}\\ \mathbf{S}\end{array} \right\rangle \rightarrow \left\langle \begin{array}{l}\mathbf{B}\\ \mathbf{Q}\{\ \ \mathsf{P1}^r_{i\rightarrow \mathrm{crd}(r)}\ \mapsto\ (b, \text{outgoing})\ \ \}\\ \mathbf{S}\{\ \ \ \ \ \ \ \ \ \ \ \ \ \ i\ \mapsto\ ((r, P), b, d)\ \ \ \}\end{array} \right\rangle} \text{ where } P \stackrel{\text{def}}{=} \begin{cases}\mathsf{P2} & \text{if } i = \mathrm{crd}(r)\\ \mathsf{P3} & \text{if } i \neq \mathrm{crd}(r)\end{cases}$$

$$\text{(PHS-2)}\ \frac{\mathbf{S}(i) = ((r, \mathsf{P2}), b, d) \qquad |\,\text{received}(\mathbf{Q}).\mathsf{P1}^r| \geq \lceil (n{+}1)/2 \rceil}{\left\langle \begin{array}{l}\mathbf{B}\\ \mathbf{Q}\\ \mathbf{S}\end{array} \right\rangle \rightarrow \left\langle \begin{array}{l}\mathbf{B}\\ \mathbf{Q}\{\ \ \mathsf{P2}^r_{\mathrm{crd}(r)\rightarrow j}\ \mapsto\ (\text{best}^r(\mathbf{Q}), \text{outgoing})\ \ \}_{\forall 1\leq j\leq n}\\ \mathbf{S}\{\ \ \ \ \ \ \ \ \ \ \ \ \ \ i\ \mapsto\ ((r, \mathsf{P3}), b, d)\ \ \ \ \ \ \ \ \}\end{array} \right\rangle}$$

$$\text{(PHS-3-TRUST)}\ \frac{\mathbf{S}(i) = ((r, \mathsf{P3}), b, d) \qquad \mathbf{Q}.\mathsf{P2}^r_{\mathrm{crd}(r)\rightarrow i} = ((v, s), \text{received})}{\left\langle \begin{array}{l}\mathbf{B}\\ \mathbf{Q}\\ \mathbf{S}\end{array} \right\rangle \rightarrow \left\langle \begin{array}{l}\mathbf{B}\\ \mathbf{Q}\{\ \ \mathsf{P3}^r_{i\rightarrow \mathrm{crd}(r)}\ \mapsto\ (\text{ack}, \text{outgoing})\ \ \}\\ \mathbf{S}\{\ \ \ \ \ \ \ \ \ \ \ \ \ \ i\ \mapsto\ ((r, P), (v, r), d)\ \ \}\end{array} \right\rangle} \text{ where } P \stackrel{\text{def}}{=} \begin{cases}\mathsf{P4} & \text{if } i = \mathrm{crd}(r)\\ \mathsf{W} & \text{if } i \neq \mathrm{crd}(r)\end{cases}$$

$$\text{(PHS-3-SUSPECT)}\ \frac{\begin{array}{r}\mathbf{S}(i)\ =\ ((r, \mathsf{P3}), b, d)\\ i\ \neq\ \mathrm{crd}(r)\end{array} \qquad \mathbf{Q}.\mathsf{P2}^r_{\mathrm{crd}(r)\rightarrow i} \neq (\cdot, \text{received})}{\left\langle \begin{array}{l}\mathbf{B}\\ \mathbf{Q}\\ \mathbf{S}\end{array} \right\rangle \rightarrow \left\langle \begin{array}{l}\mathbf{B}\\ \mathbf{Q}\{\ \ \mathsf{P3}^r_{i\rightarrow \mathrm{crd}(r)}\ \mapsto\ (\text{nack}, \text{outgoing})\ \ \}\\ \mathbf{S}\{\ \ \ \ \ \ \ \ \ \ \ \ \ \ i\ \mapsto\ ((r, \mathsf{W}), b, d)\ \ \ \ \ \ \}\end{array} \right\rangle}$$

$$\text{(PHS-4-FAIL)}\ \frac{\mathbf{S}(i) = ((r, \mathsf{P4}), b, d) \qquad \begin{array}{rcl}|\,\text{received}(\mathbf{Q}).\mathsf{P3}^r| & \geq & \lceil (n{+}1)/2 \rceil\\ |\,\text{ack}(\text{received}(\mathbf{Q}).\mathsf{P3}^r)| & < & \lceil (n{+}1)/2 \rceil\end{array}}{\left\langle \begin{array}{l}\mathbf{B}\\ \mathbf{Q}\\ \mathbf{S}\end{array} \right\rangle \rightarrow \left\langle \begin{array}{l}\mathbf{B}\\ \mathbf{Q}\\ \mathbf{S}\{\ \ i\ \mapsto\ ((r, \mathsf{W}), b, d)\ \ \}\end{array} \right\rangle}$$

$$\text{(PHS-4-SUCCESS)}\ \frac{\mathbf{S}(i) = ((r, \mathsf{P4}), (v, s), d) \qquad |\,\text{ack}(\text{received}(\mathbf{Q}).\mathsf{P3}^r)| \geq \lceil (n{+}1)/2 \rceil}{\left\langle \begin{array}{l}\mathbf{B}\\ \mathbf{Q}\\ \mathbf{S}\end{array} \right\rangle \rightarrow \left\langle \begin{array}{l}\mathbf{B}\{\ \ r\ \mapsto\ (v, \mathbb{P})\ \ \ \ \ \ \ \ \ \ \ \ \ \ \}\\ \mathbf{Q}\\ \mathbf{S}\{\ \ i\ \mapsto\ ((r, \mathsf{W}), (v, s), d)\ \ \}\end{array} \right\rangle}$$

$$\text{(RBC-DELIVER)}\ \frac{\mathbf{S}(i) = (c, b, d) \qquad \begin{array}{r}\mathbf{B}(r)\ =\ (v, D)\\ i\ \in\ D\end{array}}{\left\langle \begin{array}{l}\mathbf{B}\\ \mathbf{Q}\\ \mathbf{S}\end{array} \right\rangle \rightarrow \left\langle \begin{array}{l}\mathbf{B}\{\ \ r\ \mapsto\ (v, D\backslash\{i\})\ \ \}\\ \mathbf{Q}\\ \mathbf{S}\{\ \ i\ \mapsto\ (c, b, d')\ \ \ \ \}\end{array} \right\rangle} \text{ where } d' \stackrel{\text{def}}{=} \begin{cases}(v, r) & \text{if } d = \perp\\ d & \text{if } d \neq \perp\end{cases}$$

Table 3: The Consensus algorithm

messages that have been sent to it for its current round. The execution increments the phase of the process state to P3, and creates n outgoing messages (each of them addressed to a different process) carrying the best proposal, as selected by the coordinator.

Phase P3 of Table 1 presents an `await` with two alternatives. To render these conditions in our transition system we need to introduce two different rules. Rule (PHS-3-TRUST) is the representation of P3 when received_QPP.P2$^r \neq \emptyset$. This rule can be executed only when the process is in phase P3 and the 2nd-phase message sent by the coordinator for that round has been received. The execution of (PHS-3-TRUST) has the effect of creating an outgoing positively acknowledging 3rd-phase message directed to the coordinator of the current round, and moving the process to a new phase P. Once again, the value of P depends on whether the process is the current coordinator of the round or not, and it is equal respectively to P4 or W. As explained before for (PHS-1), the rule (PHS-3-TRUST) is not the exact representation of P3 for received_QPP.P2$^r \neq \emptyset$ because it performs the check on the coordinator identity that in the pseudo code is deferred to P4. Rule (PHS-3-SUSPECT) is the representation of P3 when the boolean condition suspected(crd(r)) is true. For convenience, a coordinator is not allowed to suspect itself, so (PHS-3-SUSPECT) can be executed by any process, except the coordinator, that is in phase P3 and has not yet received its 2nd-phase message for that round. Once again, this rule is not the exact representation of P3 for received_QPP.P2$^r \neq \emptyset$ because it (implicitly) performs here the check on the coordinator identity that in the pseudo code is deferred to P4.

Phase P4 of Table 1 contains a conditional to determine whether the process is the coordinator of the current round; as we have seen this condition is already taken into account in rules (PHS-3-TRUST) and (PHS-3-SUSPECT). Both rule (PHS-4-FAIL) and rule (PHS-4-SUCCESS) can be executed only if the process has received at least a majority of 3rd-phase messages for its current round; this models the condition `await` $|\text{received\_QPP.P3}^r| \geq \lceil (n+1)/2 \rceil$. The rule (PHS-4-SUCCESS) implements the case when $|\text{ack(received\_QPP.P3}^r)| \geq \lceil (n+1)/2 \rceil$ while rule (PHS-4-FAIL) is for the opposite case. Rule (PHS-4-FAIL) can be executed if the process is in phase P4, and if it has received at least a majority of 3rd-phase messages for the current round, but only if there is *not* a majority of positive acknowledgments. The execution simply increments the phase of the process state to W, enabling a new round to start (if the conditions of rule (WHILE) are matched). On the contrary, rule (PHS-4-SUCCESS) can be executed if the process is in phase P4, and if it has received at least a majority of positively acknowledging 3rd-phase messages for the current round (this implies that the number of 3rd-phase messages received by the process in the current round is at least equal to $\lceil (n+1)/2 \rceil$). The execution of rule (PHS-4-SUCCESS) increments the phase of the process state to W, making it possible for a new round to start. Moreover, it adds to $\mathbf{B}$, in correspondence to the current round, a new element containing the belief of the process. This is the representation of RBC_BROADCAST(i, *state*.belief) of Table 1. The identity of the process that has inserted the element in $\mathbf{B}$ can be retrieved later (if needed) from the round number to which it is associated. In fact, in each round, the only process that can be in phase P4 and can therefore execute rule (PHS-4-SUCCESS) is the coordinator of the round.

Rule (RBC-DELIVER) is the representation of the `when` block in Table 1. It can be executed at any time by any process provided that there is an element in $\mathbf{B}$ that contains the process identifier. The execution of (RBC-DELIVER) subtracts the process identifier from the list of processes that still have to deliver the message. Moreover, if the process has not yet decided (d is still undefined) the execution forces the process to decide for the value carried in the delivered message.

Notation. Most derivations of computation steps $\langle \mathbf{B}, \mathbf{Q}, \mathbf{S} \rangle \rightarrow \langle \mathbf{B}', \mathbf{Q}', \mathbf{S}' \rangle$ are fully determined by (1) the source configuration $\langle \mathbf{B}, \mathbf{Q}, \mathbf{S} \rangle$, (2) the identifier i of the process that executes the action —note that each rule "touches" at most one state component— and (3) the name (RULE) of the rule that is applied. This is true for all of the rules in Table 3 (since they model a deterministic algorithm) as well as for rule (CRASH) of Table 2. The only exceptions are the QPP-rules, where the choice of the processed message is non-deterministic. Accordingly, in statements about runs, we will occasionally denote computation steps by writing

$$\langle \mathbf{B}, \mathbf{Q}, \mathbf{S} \rangle \rightarrow_{i:(\text{RULE})} \langle \mathbf{B}', \mathbf{Q}', \mathbf{S}' \rangle .$$

Sometimes, instead of the former notation for runs,

$$\langle\, \mathbf{B}_0, \mathbf{Q}_0, \mathbf{S}_0^{(v_1,\ldots,v_n)} \,\rangle \;\rightarrow^* \langle\, \mathbf{B}, \mathbf{Q}, \mathbf{S} \,\rangle \;\rightarrow^{*/\infty}$$

we use the more succinct notation $\big(\mathfrak{C}_x\big)_T^{(v_1,\ldots,v_n)}$ as an abbreviation for

$$\big(\mathfrak{C}_x\big)_{x\in T}^{(v_1,\ldots,v_n)} \qquad \text{where} \qquad \mathfrak{C}_x = \langle\, \mathbf{B}_x, \mathbf{Q}_x, \mathbf{S}_x \,\rangle$$

and T is an interval of natural numbers, i.e., either $[\![\, t_0, t_m \,]\!]$ (denoting finite runs, or prefixes thereof) or $[\![\, t_0, \infty \,)\!)$ (denoting infinite runs) for some $t_0, t_m \in \mathbb{N}$ with $t_0 \leq t_m$, such that for all $t > t_0$:

$$\mathfrak{C}_{t-1} \;\rightarrow_{i:(\textsc{Rule})}\; \mathfrak{C}_t$$

for some $i \in \mathbb{P}$ and rule (Rule). Unless specified otherwise, we assume $t_0 = 0$.

4.4 On histories, locality and distribution.

In the context of algorithm verification, it is often useful to book-keep some execution information that is needed for the proofs. This information should not be used by the processes as knowledge (i.e., it should not affect the behavior of the algorithm under study), it should rather be used to simplify the job of the prover. *History variables* have been introduced for this purpose, as an augmentation of the state space that is not accessible to the processes. (See the *Related work* section in §7 for a comparison with related approaches and the most relevant references.)

In our work, we use the components $\mathbf{B}$ and $\mathbf{Q}$ as communication media. For the proofs, not only we need to record the messages sent or received by correct processes, but we also must keep track of the messages that possibly got lost due to process crashes. To avoid the repetition of alike structures, we enrich $\mathbf{B}$ and $\mathbf{Q}$ with additional information. This information does not affect the behavior of the algorithm because processes neither need the additional information nor can they ever take advantage of it. In the following, we clarify this double role of $\mathbf{B}$ and $\mathbf{Q}$.

In $\mathbf{B}$, Rbc-messages appear as soon as they are Rbc-broadcast, while on Rbc-delivery the intended receiver checks for the presence of a relevant message in the medium. The knowledge that a Rbc-delivery of the message by some particular process has occurred remains present in $\mathbf{B}$, always leaving behind an entry of the form $(v, \cdot)$ as witness, even if all processes have Rbc-delivered. However, no process can ever see whether another process has already Rbc-delivered, so this globally available knowledge does not influence the algorithm.

With $\mathbf{Q}$ and its interpretation given in §4.1, the situation is slightly more complicated, because the component does not only play the global role of a communication medium, but also the role of message queues, namely

$$\{\, x \in \mathbf{Q} \mid x = (\cdot, \cdot, j, \cdot, \cdot, \text{outgoing}) \,\}$$

for outgoing Qpp-messages, local at the sender (here: for process j), and

$$\{\, x \in \mathbf{Q} \mid x = (\cdot, \cdot, \cdot, k, \cdot, \text{received}) \,\}$$

for received Qpp-messages, local at the receiver (here: for process k). The double use of $\mathbf{Q}$ as communication medium and message queues provides us with convenient mathematical proof structures. For example, outgoing message queues allow us to model the simultaneous sending of multiple messages by an atomic step of the algorithm. The crash-sensitive passage of individual messages from the outgoing queue to the communication medium is left to an independent rule, but still works on the same data structure by just changing message tags. As in the case of Rbc-messages, Qpp-messages are never thrown away, but are rather kept in the data structure to witness their prior existence in the run.

Despite the apparently global availability of the structure $\mathbf{Q}$, in particular of all of its queues, the execution of any rule of the algorithm by any process maintains a purely local character. In fact, whenever some process checks for the presence of one or several messages in the medium,

it always only checks for their presence in subcomponents that represent its own local message queues. Symmetrically, whenever some process sends a message, this is put in the process' own local outgoing queue.

Summing up, when considered as communication media both $\mathbf{B}$ and $\mathbf{Q}$ model the complete lifetime of the respective kinds of messages. We consider a QPP-message as *dead* in three different cases: (i) when it is stuck in the outgoing-queue of a crashed process, (ii) when it is stuck in the received-queue of a crashed process, (iii) when it is kept in a received-queue and it has already been processed, (iv) when it is in transit and its receiver has crashed.

We consider a RBC-message as *dead* if it is kept in $\mathbf{B}$ but the set of processes that still have to RBC-deliver it is either empty or contains only crashed processes. The history character of $\mathbf{B}$ and $\mathbf{Q}$ is represented precisely by the fact that they keep information about each message even beyond the death of the message itself, which is in contrast to the mere algorithmic use. This kind of historical information would not be as easily accessible, and thus not as easy to model, if we did not integrate the multiple local QPP-messages queues within the single global component $\mathbf{Q}$.

4.5 Formal specification of the underlying abstractions

According to [CT96], a process is called *correct* in a given run, if it does not crash in that run. To formalize this notion in our model, we just need to check the definedness of the relevant state entry in all configurations of the given run.

Definition 4.5.1 (Correct Processes) *Let* $R = \big(\langle\, \mathbf{B}_x, \mathbf{Q}_x, \mathbf{S}_x \,\rangle\big)_T^{(v_1,\ldots,v_n)}$. *We define*

$$\mathsf{Correct}(R) := \{\, i \in \mathbb{P} \mid \forall t \in T : \mathbf{S}_t(i) \neq \bot \,\}\ .$$

A process $i \in \mathsf{Correct}(R)$ *is called* correct in R.

In §2.1, we informally introduced the notion of QPP-messaging. To formally state the quasi-reliability property of the messaging service based on our model, we need to specify when precisely the message of interest appears in a run for the first time, i.e., when the intended primitive QPP_SEND is executed. There are several rules in our semantics that start the sending of a message. In contrast, there is only a single rule that corresponds to QPP_RECEIVE, namely rule (QPP-DELIVER).

Definition 4.5.2 (Quasi-Reliable Point-To-Point) *Let* $R = \big(\langle\, \mathbf{B}_x, \mathbf{Q}_x, \mathbf{S}_x \,\rangle\big)_T^{(v_1,\ldots,v_n)}$ *denote a run. R satisfies* QPP-Reliability *if, for all processes* $j, k \in \mathsf{Correct}(R)$ *and time* $t > 0$, *whenever*

$$\langle\, \cdot, \mathbf{Q}_{t-1}, \cdot \,\rangle \ \rightarrow_{j:(\cdot)}\ \langle\, \cdot, \mathbf{Q}_t, \cdot \,\rangle$$

with $\mathbf{Q}_{t-1}.P^r_{j\to k} = \bot$ *and* $\mathbf{Q}_t.P^r_{j\to k} = (\cdot, \text{outgoing})$, *for* $r > 0$ *and* $P \in \{\, \mathsf{P1}, \mathsf{P2}, \mathsf{P3} \,\}$, *then there is* $u > t$ *such that*

$$\langle\, \cdot, \mathbf{Q}_{u-1}, \cdot \,\rangle \ \rightarrow_{k:(\text{QPP-DELIVER})}\ \langle\, \cdot, \mathbf{Q}_u, \cdot \,\rangle$$

with $\mathbf{Q}_u.P^r_{j\to k} = (\cdot, \text{received})$.

It turns out that, by construction, and because we do not have message loss, finite complete Consensus runs are always QPP-reliable in the sense that all QPP-messages exchanged between correct processes must have been received when the final state is reached.

In §2.2, we informally introduced the notion of the FD Ω, which is defined to satisfy the following property: there is a time $\hat{t}$ after which all processes always trust (i.e., do not suspect) the same correct process k. Now, we formalize this property in our rewrite system by expressing that, whenever after time $\hat{t}$ some process i suspects another process—necessarily the coordinator of the round r that the suspecting process i currently visits—then this suspected process is different from the trusted process k.

Definition 4.5.3 (Failure Detection) *Let* $R = \big(\langle \mathbf{B}_x, \mathbf{Q}_x, \mathbf{S}_x \rangle\big)_T^{(v_1,\ldots,v_n)}$ *be a run. R satisfies the property Ω if there is $k \in \mathsf{Correct}(R)$ and there is $\hat{t} > 0$ such that for all $t > \hat{t}$, whenever*

$$\langle \cdot, \cdot, \mathbf{S}_{t-1} \rangle \rightarrow_{i:(\text{PHS-3-SUSPECT})} \langle \cdot, \cdot, \mathbf{S}_t \rangle$$

and $\mathbf{S}_t(i) = ((r, P), \cdot, \cdot)$*, then* $k \neq \mathrm{crd}(r)$*.*

Note that this definition does not explicitly model the trust in process k via some (global) mathematical structure (as the run-specific time-dependent function $H_\Omega : \mathbb{P} \times T \rightarrow \mathbb{P}$ of [CHT96]), but rather just observes that process k *is* obviously trusted in a given run. Since the explicit modeling is used for precisely the purpose of detecting a trusted process, we consider the two representations as being equivalent.

In § 2.3, we informally introduced the notion of Reliable Broadcast. For the formal definition of the properties of Reliable Broadcast we have a direct correspondence between the informal terminology of "executing a primitive" and the execution of a particular transition rule. In order to make precise that some RBC-delivery actually corresponds to a particular RBC-broadcast, we explicitly refer to the round r of the respective (preceding) execution of broadcast.

Definition 4.5.4 (Reliable Broadcast) *Let* $R = \big(\langle \mathbf{B}_x, \mathbf{Q}_x, \mathbf{S}_x \rangle\big)_T^{(v_1,\ldots,v_n)}$ *denote a run. Then we define some of its properties:*

1. RBC-Validity: *for every $i \in \mathsf{Correct}(R)$, if there is $t > 0$ such that*

   $$\langle \mathbf{B}_{t-1}, \cdot, \cdot \rangle \rightarrow_{i:(\text{PHS-4-SUCCESS})} \langle \mathbf{B}_t, \cdot, \cdot \rangle$$

   with $\mathbf{B}_{t-1}(r) = \perp$ and $\mathbf{B}_t(r) = (v, \mathbb{P})$ for some $r > 0$, then there is $u > t$ such that

   $$\langle \mathbf{B}_{u-1}, \cdot, \cdot \rangle \rightarrow_{i:(\text{RBC-DELIVER})} \langle \mathbf{B}_u, \cdot, \cdot \rangle$$

   with $\mathbf{B}_{u-1}(r) = (v, D_{u-1})$, $\mathbf{B}_u(r) = (v, D_u)$ and $D_{u-1} = D_u \uplus \{i\}$.
2. RBC-Agreement: *for every $i \in \mathsf{Correct}(R)$, if there is $t > 0$ such that*

   $$\langle \mathbf{B}_{t-1}, \cdot, \cdot \rangle \rightarrow_{i:(\text{RBC-DELIVER})} \langle \mathbf{B}_t, \cdot, \cdot \rangle$$

   with $\mathbf{B}_{t-1}(r) = (v, D_{t-1})$, $\mathbf{B}_t(r) = (v, D_t)$ and $D_{t-1} = D_t \uplus \{i\}$ for some $r > 0$, then, for every $j \in \mathsf{Correct}(R)$, there is $u > 0$ such that

   $$\langle \mathbf{B}_{u-1}, \cdot, \cdot \rangle \rightarrow_{j:(\text{RBC-DELIVER})} \langle \mathbf{B}_u, \cdot, \cdot \rangle$$

   with $\mathbf{B}_{u-1}(r) = (v, D_{u-1})$, $\mathbf{B}_u(r) = (v, D_u)$ and $D_{u-1} = D_u \uplus \{j\}$.
3. (Uniform) RBC-Integrity: *for every $i \in \mathsf{Correct}(R)$, if there are $t_1, t_2 > 0$ such that*

   $$\langle \mathbf{B}_{t_1-1}, \cdot, \cdot \rangle \rightarrow_{i:(\text{RBC-DELIVER})} \langle \mathbf{B}_{t_1}, \cdot, \cdot \rangle \qquad \langle \mathbf{B}_{t_2-1}, \cdot, \cdot \rangle \rightarrow_{i:(\text{RBC-DELIVER})} \langle \mathbf{B}_{t_2}, \cdot, \cdot \rangle$$

   and, for some $r > 0$, both

   (a) $\mathbf{B}_{t_1-1}(r) = (v, D_{t_1-1})$, $\mathbf{B}_{t_1}(r) = (v, D_{t_1})$, and $D_{t_1-1} = D_{t_1} \uplus \{i\}$.
   (b) $\mathbf{B}_{t_2-1}(r) = (v, D_{t_2-1})$, $\mathbf{B}_{t_2}(r) = (v, D_{t_2})$, and $D_{t_2-1} = D_{t_2} \uplus \{i\}$.

   then $t_1 = t_2$ and there is $u < t_1$ such that

   $$\langle \mathbf{B}_{u-1}, \cdot, \cdot \rangle \rightarrow_{i:(\text{PHS-4-SUCCESS})} \langle \mathbf{B}_u, \cdot, \cdot \rangle$$

   with $\mathbf{B}_{u-1}(r) = \perp$ and $\mathbf{B}_u(r) = (v, \mathbb{P})$.

Finally, we put all the properties together to select those runs of Consensus where only a minority of processes may crash, and that are based on the proper functioning of both the communication and the failure detection services.

Definition 4.5.5 (Admissible runs) *A Consensus run* $R = \big(\langle \mathbf{B}_x, \mathbf{Q}_x, \mathbf{S}_x \rangle\big)_T^{(v_1,\ldots,v_n)}$ *is called* admissible *if it satisfies* QPP*-Reliability, the three* RBC*-properties, the Ω-property and, in addition, exhibits a majority of correct processes, i.e.,* $\mathsf{Correct}(R) \geq \lceil (n+1)/2 \rceil$*.*

We will prove that all admissible runs of Consensus satisfy Validity, Agreement and Termination.

5 Basic properties of the algorithm

In this section we prove a number of basic properties of the Consensus algorithm defined in Section 4.3. Most of them are simple, but still very useful properties holding for (prefixes of) runs of the form:

$$\langle\, \mathbf{B}_0, \mathbf{Q}_0, \mathbf{S}_0^{(v_1,\ldots,v_n)} \,\rangle \;\rightarrow^* \langle\, \mathbf{B}, \mathbf{Q}, \mathbf{S} \,\rangle\ .$$

For the sake of clarity, we give in this section only the most significant proofs; we relegate the remaining ones to the Appendix. When carrying out proofs more informally, on the basis of pseudo code, the properties of this section are those that would typically be introduced via the words "by inspection of the code". To express them formally, we first need to define some appropriate orderings on the components of configurations.

5.1 Orderings

Natural orders can be defined on process states as well as point-to-point and broadcast histories.

Phase identifiers are transitively ordered as $\mathsf{P1} < \mathsf{P2} < \mathsf{P3} < \mathsf{P4} < \mathsf{W}$; we write $\leq$ to denote the reflexive closure of $<$. From the order on phases it is easy to derive a partial order on program counters that takes into account whether the respective process, say i, is coordinator of a particular round r.

Definition 5.1.1 (Ordering Program Counters) *Let $P, P' \in \{\, \mathsf{P1}, \mathsf{P2}, \mathsf{P3}, \mathsf{P4}, \mathsf{W} \,\}$ and $r, r' \in \mathbb{N}$.*

1. *We write $(r, P) \leq (r', P')$ if $r < r'$ or $(r = r' \wedge P \leq P')$.*
2. *The pair (r, P) is a* valid program counter *for some process i if:*
   (a) *$i \neq \mathrm{crd}(r)$ implies $P \in \{\, \mathsf{P1}, \mathsf{P3}, \mathsf{W} \,\}$, and*
   (b) *$r = 0$ implies $P = \mathsf{W}$.*
3. *We write $(r, P) \leq_i (r', P')$ if $(r, P) \leq (r', P')$ and both (r, P) and (r', P') are valid for process i.*

We write $(r, P) <_i (r', P')$ (respectively, $(r, P) < (r', P')$) if $(r, P) \leq_i (r', P')$ (respectively, $(r, P) \leq (r', P')$) and $(r, P) \neq (r', P')$.

Notice that in the definition of $\leq_i$ the validity conditions are necessary as only coordinators go through phases $\mathsf{P2}$ and $\mathsf{P4}$, and as round 0 starts in phase W. Due to the latter, there is no r and P such that $(r, P) <_i (0, \mathsf{W})$.

If we consider the partial order on program counters, together with the information on the processes' crash status (where defined is smaller than undefined) and the value of the decision field (where undefined is smaller than defined), then we can establish a partial order on both the global state array and the local states of the processes.

Definition 5.1.2 (Ordering States) *Let $\mathbf{S}$ and $\mathbf{S}'$ be process states and $i \in \mathbb{P}$. We write $\mathbf{S}(i) \leq \mathbf{S}'(i)$ if*

1. *either $\mathbf{S}'(i) = \perp$,*
2. *or $\mathbf{S}(i){=}(c_i, \cdot, d_i)$ and $\mathbf{S}'(i){=}(c'_i, \cdot, d'_i)$ where*
   - *either $c_i <_i c'_i$,*
   - *or $c_i = c'_i \;\wedge\; ((d_i = d'_i) \vee d_i = \perp)$.*

We write $\mathbf{S} \leq \mathbf{S}'$ if for all $i \in \mathbb{P}$ it holds that $\mathbf{S}(i) \leq \mathbf{S}'(i)$. We write $\mathbf{S}(i) < \mathbf{S}'(i)$ if $\mathbf{S}(i) \leq \mathbf{S}'(i)$ and $\mathbf{S}(i) \neq \mathbf{S}'(i)$. Similarly, we write $\mathbf{S} < \mathbf{S}'$ if $\mathbf{S} \leq \mathbf{S}'$ and $\mathbf{S} \neq \mathbf{S}'$.

Note that the global ordering on state arrays properly reflects the local character of the transition rules of §4.3. As we will show with Lemma 5.2.1 each computation step of the algorithm corresponds precisely to the state change of a single process.

Message histories are ordered as follows: an entry defined in the later history must either be undefined in the older history, or its tag in the older history is smaller according to the transitive order:

$$\text{outgoing} < \text{transit} < \text{received} .$$

Definition 5.1.3 (Ordering QPP-Message Histories) *Let $\mathbf{Q}, \mathbf{Q}'$ be QPP-message histories. Then $\mathbf{Q} \leq \mathbf{Q}'$ iff for all $X \in \{ \mathsf{P1}^r_{i \to \mathrm{crd}(r)}, \mathsf{P2}^r_{\mathrm{crd}(r) \to i}, \mathsf{P3}^r_{i \to \mathrm{crd}(r)} \}$, with $i \in \mathbb{P}$ and $r \in \mathbb{N}$:*

1. *either* $\mathbf{Q}.X = \bot$
2. *or* $\mathbf{Q}.X = (V, \tau) \ \wedge \ \mathbf{Q}'.X = (V, \tau') \ \wedge \ \tau \leq \tau'$.

As usual, we define the strict counterpart as $\mathbf{Q} < \mathbf{Q}'$ iff $\mathbf{Q} \leq \mathbf{Q}'$ and $\mathbf{Q} \neq \mathbf{Q}'$.

In fact, our transition rules (QPP_SND) and (QPP_DELIVER) overwrite transmission tags, but only in increasing order while leaving the other data unchanged (see Corollary 5.2.5). Note that the order of QPP-message histories is transitive.

Finally, broadcast histories are ordered according to the decreasing delivery set of their entries.

Definition 5.1.4 (Ordering RBC-Message Histories) *Let $\mathbf{B}, \mathbf{B}'$ be RBC-message histories and $r \in \mathbb{N}$. We write $\mathbf{B}(r) \leq \mathbf{B}'(r)$ iff*

1. *either $\mathbf{B}(r) = \bot$*
2. *or $\mathbf{B}(r) = (v, D)$ and $\mathbf{B}'(r) = (v, D')$ with $D \supseteq D'$.*

We write $\mathbf{B} \leq \mathbf{B}'$ if for all $r \in \mathbb{N}$ it holds that $\mathbf{B}(r) \leq \mathbf{B}'(r)$. We write $\mathbf{B}(r) < \mathbf{B}'(r)$ if $\mathbf{B}(r) \leq \mathbf{B}'(r)$ and $\mathbf{B}(r) \neq \mathbf{B}'(r)$. Similarly, we write $\mathbf{B} < \mathbf{B}'$ if $\mathbf{B} \leq \mathbf{B}'$ and $\mathbf{B} \neq \mathbf{B}'$.

As for QPP-message histories, overwriting of RBC-message histories may happen, but only decreasing the delivery set, while keeping the recorded broadcast value unchanged (see Lemma 5.2.4).

5.2 Monotonicity and message evolution

In every Consensus run, states and message histories are monotonically non-decreasing. We start from the monotonicity of the state component, which is derived from the respective program counters.

Lemma 5.2.1 (Monotonicity of States) *Let $R = \big(\langle \mathbf{B}_x, \mathbf{Q}_x, \mathbf{S}_x \rangle \big)_T^{(v_1, \ldots, v_n)}$ be a run. Let*

$$\langle \mathbf{B}_{u-1}, \mathbf{Q}_{u-1}, \mathbf{S}_{u-1} \rangle \ \rightarrow_{i:(\textsc{Rule})} \langle \mathbf{B}_u, \mathbf{Q}_u, \mathbf{S}_u \rangle$$

be an arbitrary computation step of R, for some $u \in T$, $i \in \mathbb{P}$, with $\mathbf{S}_{u-1}(i) = (c_{u-1}, \cdot, \cdot)$.

1. *If* (RULE) *is rule* (CRASH)*, then $\mathbf{S}_u = \bot$, and hence $\mathbf{S}_{u-1}(i) < \mathbf{S}_u(i)$ and $\mathbf{S}_{u-1} < \mathbf{S}_u$.*
2. *If* (RULE) *is any of the rules* (QPP_SEND) *or* (QPP_DELIVER)*, then $\mathbf{S}_u(i) = (c_u, \cdot, \cdot)$ with $c_{u-1} = c_u$, and hence $\mathbf{S}_{u-1}(i) = \mathbf{S}_u(i)$, and $\mathbf{S}_{u-1} = \mathbf{S}_u$.*
3. *If* (RULE) *is rule* (WHILE)*, then $\mathbf{S}_u(i) = (c_u, \cdot, \cdot)$ with $c_{u-1} < c_u$, and hence $\mathbf{S}_{u-1}(i) < \mathbf{S}_u(i)$, and $\mathbf{S}_{u-1} < \mathbf{S}_u$.*
4. *If* (RULE) *is any of the* (PHS-$*$)*-rules, then $\mathbf{S}_u(i) = (c_u, \cdot, \cdot)$ with $c_{u-1} < c_u$, and hence $\mathbf{S}_{u-1}(i) < \mathbf{S}_u(i)$, and $\mathbf{S}_{u-1} < \mathbf{S}_u$.*
5. *If* (RULE) *is rule* (RBC-DELIVER)*, then $\mathbf{S}_u(i) = (c_u, \cdot, \cdot)$ with $c_{u-1} = c_u$, and hence $\mathbf{S}_{u-1}(i) \leq \mathbf{S}_u(i)$, and $\mathbf{S}_{u-1} \leq \mathbf{S}_u$.*

Proof. Immediate from the local comparison of premises and conclusions of each individual rule (RULE), and Definition 5.1.2 of the state ordering derived from program counters. □

Proving the monotonicity of the QPP-message history $\mathbf{Q}$ and of the RBC-message history $\mathbf{B}$ takes instead a bit more effort (see Lemma 5.2.4). Analyzing the transition rules in Table 2 and 3, we see that there are precisely four rules that *add* messages to a component $\mathbf{Q}$, namely (PHS-1),

(PHS-2), (PHS-3-TRUST), and (PHS-3-SUSPECT). Moreover, there are two rules that change the content of messages in a component $\mathbf{Q}$, namely (QPP_SND) and (QPP_DELIVER). All the other rules leave the component $\mathbf{Q}$ untouched. As for $\mathbf{B}$, there is only one rule that adds messages to it (PHS-4-SUCCESS). Moreover, the application of rule (RBC-DELIVER) changes the content of messages in a component $\mathbf{B}$ while all other rules leave $\mathbf{B}$ untouched.

The following lemma describes how 1st-, 2nd-, 3rd-phase and broadcast messages are created, essentially by reading the transition rules backwards and identifying appropriate conditions. More precisely, from the definedness of an entry in the QPP-message or in the RBC-message history of some reachable configuration, we can derive that there must have been along the way (starting from the initial configuration) a uniquely defined computation step, where this particular entry was first added. This addition must have been produced by one of the aforementioned five rules. In order to uniquely identify such steps, we decompose the run that led to the assumed reachable configuration into four parts, as follows. (We recall here that with $\mathbf{Q}_u.P^r_{i\to j} \neq \bot$ we mean that at time u the entry for message $P^r_{i\to j}$ is defined and has the form $(\mathit{contents}, \tau)$. This means that at time u, message $P^r_{i\to j}$ has been sent.)

Lemma 5.2.2 (Message Creation) *Let $R = \big(\langle\, \mathbf{B}_x, \mathbf{Q}_x, \mathbf{S}_x \,\rangle\big)_T^{(v_1,\ldots,v_n)}$ be a run. Then,*

1. *If $\mathbf{Q}_u.\mathsf{P1}^r_{i\to \mathrm{crd}(r)} \neq \bot$, for some $u \in T$, $i \in \mathbb{P}$, and $r > 0$, then there is $t \in T$, $t \leq u$, for which*
$$\langle\, \mathbf{B}_{t-1}, \mathbf{Q}_{t-1}, \mathbf{S}_{t-1} \,\rangle \;\to_{i:(\text{PHS-1})}\; \langle\, \mathbf{B}_t, \mathbf{Q}_t, \mathbf{S}_t \,\rangle$$
such that, for some $b \in \mathbb{V} \times \mathbb{N}$
$$\begin{array}{rcl@{\qquad}rcl} \mathbf{Q}_{t-1}.\mathsf{P1}^r_{i\to \mathrm{crd}(r)} & = & \bot & \mathbf{S}_{t-1}(i) & = & \big((r,\mathsf{P1}), b, \cdot\big) \\ \mathbf{Q}_t.\mathsf{P1}^r_{i\to \mathrm{crd}(r)} & = & (b, \text{outgoing}) & \mathbf{S}_t(i) & = & \big((r,P), b, \cdot\big) \end{array}$$
where, if $i = \mathrm{crd}(r)$ then $P = \mathsf{P2}$, otherwise $P = \mathsf{P3}$.
2. *If $\mathbf{Q}_u.\mathsf{P2}^r_{\mathrm{crd}(r)\to k} \neq \bot$, for some $u \in T$, $k \in \mathbb{P}$, and $r > 0$, then there is $t \in T$, $t \leq u$, for which*
$$\langle\, \mathbf{B}_{t-1}, \mathbf{Q}_{t-1}, \mathbf{S}_{t-1} \,\rangle \;\to_{\mathrm{crd}(r):(\text{PHS-2})}\; \langle\, \mathbf{B}_t, \mathbf{Q}_t, \mathbf{S}_t \,\rangle$$
such that, for some $b \in \mathbb{V} \times \mathbb{N}$ and for all $i \in \mathbb{P}$
$$\begin{array}{rcl@{\qquad}rcl} \mathbf{Q}_{t-1}.\mathsf{P2}^r_{\mathrm{crd}(r)\to i} & = & \bot & \mathbf{S}_{t-1}(\mathrm{crd}(r)) & = & \big((r,\mathsf{P2}), b, \cdot\big) \\ \mathbf{Q}_t.\mathsf{P2}^r_{\mathrm{crd}(r)\to i} & = & (\mathrm{best}^r(\mathbf{Q}_{t-1}), \text{outgoing}) & \mathbf{S}_t(\mathrm{crd}(r)) & = & \big((r,\mathsf{P3}), b, \cdot\big) \end{array}$$
3. *If $\mathbf{Q}_u.\mathsf{P3}^r_{i\to \mathrm{crd}(r)} \neq \bot$, for some $u \in T$, $i \in \mathbb{P}$, and $r > 0$, then there is $t \in T$, $t \leq u$, for which*
$$\langle\, \mathbf{B}_{t-1}, \mathbf{Q}_{t-1}, \mathbf{S}_{t-1} \,\rangle \;\to_{i:(\text{RULE})}\; \langle\, \mathbf{B}_t, \mathbf{Q}_t, \mathbf{S}_t \,\rangle$$
such that either of the following is true, for some $b \in \mathbb{V} \times \mathbb{N}$:
   (a) *Rule* (RULE) *is* (PHS-3-SUSPECT), *$i \neq \mathrm{crd}(r)$ and*
$$\begin{array}{rcl@{\qquad}rcl} \mathbf{Q}_{t-1}.\mathsf{P3}^r_{i\to \mathrm{crd}(r)} & = & \bot & \mathbf{S}_{t-1}(i) & = & \big((r,\mathsf{P3}), b, \cdot\big) \\ \mathbf{Q}_t.\mathsf{P3}^r_{i\to \mathrm{crd}(r)} & = & (\text{nack}, \text{outgoing}) & \mathbf{S}_t(i) & = & \big((r,\mathsf{W}), b, \cdot\big) \end{array}$$
   (b) *Rule* (RULE) *is* (PHS-3-TRUST) *and*
$$\begin{array}{rcl@{\qquad}rcl} \mathbf{Q}_{t-1}.\mathsf{P3}^r_{i\to \mathrm{crd}(r)} & = & \bot & \mathbf{S}_{t-1}(i) & = & \big((r,\mathsf{P3}), b, \cdot\big) \\ \mathbf{Q}_t.\mathsf{P3}^r_{i\to \mathrm{crd}(r)} & = & (\text{ack}, \text{outgoing}) & \mathbf{S}_t(i) & = & \big((r,P), (\mathrm{val}^r(\mathbf{Q}_{t-1}), r), \cdot\big) \end{array}$$
   where $\mathrm{val}^r(\mathbf{Q}_{t-1}) \neq \bot$ and if $i = \mathrm{crd}(r)$ then $P = \mathsf{P4}$, otherwise $P = \mathsf{W}$.
4. *If $\mathbf{B}_u(r) \neq \bot$, for some $u \in T$ and for some $r > 0$, then there is $t \in T$, $t \leq u$, for which*
$$\langle\, \mathbf{B}_{t-1}, \mathbf{Q}_{t-1}, \mathbf{S}_{t-1} \,\rangle \;\to_{\mathrm{crd}(r):(\text{PHS-4-SUCCESS})}\; \langle\, \mathbf{B}_t, \mathbf{Q}_t, \mathbf{S}_t \,\rangle$$
such that, for some $v \in \mathbb{V}$
$$\begin{array}{rcl@{\qquad}rcl} \mathbf{B}_{t-1}(r) & = & \bot & \mathbf{S}_{t-1}(\mathrm{crd}(r)) & = & \big((r,\mathsf{P4}), (v,\cdot), \cdot\big) \\ \mathbf{B}_t(r) & = & (v, \mathbb{P}) & \mathbf{S}_t(\mathrm{crd}(r)) & = & \big((r,\mathsf{W}), (v,\cdot), \cdot\big)\ . \end{array}$$

An important property of the Consensus algorithm is that both QPP- and RBC-messages are unique. This is guaranteed by the standard technique of adding enough distinguishing information to messages. Intuitively, we formalize the absence of message duplication by stating that *every addition of a message is fresh*. More precisely, whenever in a run a computation step is derived using one of the five rules that *add* a message to some component $\mathbf{Q}$ or $\mathbf{B}$, then the respective entry *must* have been undefined up to this very moment. An alternative and slightly more technical interpretation of this result is: if we had included conditions to ensure the prior non-definedness of added messages as premises to the respective rules, then those conditions would have been redundant.

Lemma 5.2.3 (Absence of Message Duplication) *Let* $R = \big(\langle\, \mathbf{B}_x, \mathbf{Q}_x, \mathbf{S}_x \,\rangle\big)_T^{(v_1,\ldots,v_n)}$ *be a run such that, for some* $u \in T$*,*

$$\langle\, \mathbf{B}_{u-1}, \mathbf{Q}_{u-1}, \mathbf{S}_{u-1} \,\rangle \;\rightarrow_{i:(\textsc{Rule})}\; \langle\, \mathbf{B}_u, \mathbf{Q}_u, \mathbf{S}_u \,\rangle$$

with $\mathbf{S}_{u-1}(i) = ((r, P), \cdot, \cdot)$*, for some* $r \in \mathbb{N}$*,* $r > 0$*, and* $P \in \{\mathsf{P1}, \mathsf{P2}, \mathsf{P3}, \mathsf{P4}\}$*. Then,*

1. *If* (RULE) *is* (PHS-1)*, then* $\mathbf{Q}_{u-1}.\mathsf{P1}^r_{i\rightarrow \mathrm{crd}(r)} = \bot$*.*
2. *If* (RULE) *is* (PHS-2)*, then* $\mathbf{Q}_{u-1}.\mathsf{P2}^r_{\mathrm{crd}(r)\rightarrow k} = \bot$*.*
3. *If* (RULE) *is* (PHS-3-TRUST) *or* (PHS-3-SUSPECT)*, then* $\mathbf{Q}_{u-1}.\mathsf{P3}^r_{i\rightarrow \mathrm{crd}(r)} = \bot$*.*
4. *If* (RULE) *is* (PHS-4-SUCCESS)*, then* $\mathbf{B}_{u-1}(r) = \bot$*.*

Putting it all together, we can now state the monotonicity results for the two kinds of messages.

Lemma 5.2.4 (Monotonicity of Message Histories) *Let* $R = \big(\langle\, \mathbf{B}_x, \mathbf{Q}_x, \mathbf{S}_x \,\rangle\big)_T^{(v_1,\ldots,v_n)}$ *be a run such that*

$$\langle\, \mathbf{B}_{u-1}, \mathbf{Q}_{u-1}, \mathbf{S}_{u-1} \,\rangle \;\rightarrow_{i:(\textsc{Rule})}\; \langle\, \mathbf{B}_u, \mathbf{Q}_u, \mathbf{S}_u \,\rangle$$

for some $u \in T$ *and* $i \in \mathbb{P}$*.*

1. *If* (RULE) *is any of the rules* (WHILE)*,* (PHS-4-FAIL)*,* (PHS-4-SUCCESS)*,* (RBC-DELIVER)*,* (CRASH)*, then* $\mathbf{Q}_{u-1} = \mathbf{Q}_u$*.*
2. *If* (RULE) *is any of the rules* (QPP_SND)*,* (QPP_DELIVER)*,* (PHS-1)*,* (PHS-2)*,* (PHS-3-TRUST) *or* (PHS-3-SUSPECT)*. then* $\mathbf{Q}_{u-1} < \mathbf{Q}_u$*.*
3. *If* (RULE) *is any of the rules* (PHS-4-SUCCESS) *or* (RBC-DELIVER)*, then* $\mathbf{B}_{u-1} < \mathbf{B}_u$*.*
4. *If* (RULE) *is any of the rules* (WHILE)*,* (PHS-1)*,* (PHS-2)*,* (PHS-3-TRUST)*,* (PHS-3-SUSPECT)*,* (PHS-4-FAIL)*,* (QPP_SND)*,* (QPP_DELIVER)*, or* (CRASH)*, then* $\mathbf{B}_{u-1} = \mathbf{B}_u$*.*

Proof. We prove the result by local comparison of premises and conclusions of each possible rule (RULE).
If (RULE) is any of the rules (WHILE), (PHS-4-FAIL), (PHS-4-SUCCESS), (RBC-DELIVER), (CRASH) the $\mathbf{Q}$ component is not touched, and $\mathbf{Q}_{u-1} = \mathbf{Q}_u$.
Similarly, if (RULE) is any of the rules (WHILE), (PHS-1), (PHS-2), (PHS-3-TRUST), (PHS-3-SUSPECT), (PHS-4-FAIL), (QPP_SND), (QPP_DELIVER), or (CRASH), then the $\mathbf{B}$ component is not touched, and $\mathbf{B}_{u-1} = \mathbf{B}_u$.
If (RULE) is (PHS-1), (PHS-2), (PHS-3-TRUST), or (PHS-3-SUSPECT), then we know (Lemma 5.2.3) that $\mathbf{Q}_{u-1}.X = \bot$, while the application of (RULE) produces $\mathbf{Q}_u.X \neq \bot$ with, respectively, $X \in \{\, \mathsf{P1}^r_{j\rightarrow k}, \mathsf{P2}^r_{j\rightarrow k}, \mathsf{P3}^r_{j\rightarrow k} \,\}$. Since $\mathbf{Q}_{u-1} \neq \mathbf{Q}_u$, by Definition 5.1.3 we can say that $\mathbf{Q}_{u-1} < \mathbf{Q}_u$.
If (RULE) is (QPP_SND) or (QPP_DELIVER), then the message tag is increased and by Definition 5.1.3 we can conclude $\mathbf{Q}_{u-1} < \mathbf{Q}_u$.
If (RULE) is (PHS-4-SUCCESS) or (RBC-DELIVER), then a new (Lemma 5.2.3) message is broadcasted or an already existing message is delivered. By Definition 5.1.4 we have that $\mathbf{B}_{u-1} \leq \mathbf{B}_u$, and, since $\mathbf{B}_{u-1} \neq \mathbf{B}_u$, we conclude $\mathbf{B}_{u-1} < \mathbf{B}_u$. □

Finally, we give here two straightforward results on the monotonicity of message histories. The first states that the *contents* of messages that are stored within $\mathbf{Q}$ never change; only the transmission *tag* may change. The second states that round proposals never get overwritten by later updates of $\mathbf{Q}$.

Corollary 5.2.5 *Let* $R = \big(\langle\, \mathbf{B}_x, \mathbf{Q}_x, \mathbf{S}_x \,\rangle\big)_T^{(v_1,\ldots,v_n)}$ *be a run and* $X \in \{\, \mathsf{P1}^r_{j\to k}, \mathsf{P2}^r_{j\to k}, \mathsf{P3}^r_{j\to k} \,\}$, *where* $r \in \mathbb{N}$ *and* $j, k \in \mathbb{P}$. *Let* $u, w \in T$ *with* $u \le w$.

1. *If* $\mathbf{Q}_u.X = (V, \tau)$, *then there is* τ' *such that* $\mathbf{Q}_w.X = (V, \tau')$ *with* $\tau \le \tau'$.
2. *If* $\mathbf{B}_u = (v, D)$, *then there is* D' *such that* $\mathbf{B}_w = (v, D')$ *with* $D \supseteq D'$.

Proof.

1. By Definition 5.1.3 and iterated application of Lemma 5.2.4.
2. By Definition 5.1.4 and iterated application of Lemma 5.2.4.

□

Corollary 5.2.6 *Let* $R = \big(\langle\, \mathbf{B}_x, \mathbf{Q}_x, \mathbf{S}_x \,\rangle\big)_T^{(v_1,\ldots,v_n)}$ *be a run. Let* $\mathrm{val}^r(\mathbf{Q}_u) \neq \bot$, *for some* $u \in T$ *and round* $r > 0$, *then for all* $w \in T$, $w \ge u$, *it holds that* $\mathrm{val}^r(\mathbf{Q}_w) = \mathrm{val}^r(\mathbf{Q}_u)$.

Proof. By the definition of $\mathrm{val}^r(\mathbf{Q})$ and Corollary 5.2.5 for the case 2nd-phase messages. □

5.3 Knowledge about reachable states

In this section we prove a number of properties of the states that can be reached by the algorithm. These results will be crucial for proving Agreement and Termination.

In any Consensus run, no program counter can be skipped. More precisely, for any given process i, if this process reaches some program counter, then it must have passed through all the earlier valid program counters as well.

Lemma 5.3.1 (Program counters cannot be skipped) *Let* $R = \big(\langle\, \mathbf{B}_x, \mathbf{Q}_x, \mathbf{S}_x \,\rangle\big)_T^{(v_1,\ldots,v_n)}$ *be a run. Let* $t \in T$ *and* $i \in \mathbb{P}$ *such that* $\mathbf{S}_t(i) = \big((r_t, P_t), \cdot, \cdot\big)$. *Then, for all valid program counters* (r, P) *with* $(0, \mathsf{W}) \le_i (r, P) <_i (r_t, P_t)$, *there is* $u \in T$, $u < t$, *such that* $\mathbf{S}_u(i) = \big((r, P), \cdot, \cdot\big)$.

Proof. By induction on $t \in T$, corresponding to the length of

$$\langle\, \mathbf{B}_0, \mathbf{Q}_0, \mathbf{S}_0^{(v_1,\ldots,v_n)} \,\rangle \;\to^t \langle\, \mathbf{B}_t, \mathbf{Q}_t, \mathbf{S}_t \,\rangle.$$

Base case $t = 0$ **:** By definition, for all $i \in \mathbb{P}$, we have $\mathbf{S}_t(i) = \mathbf{S}_0^{(v_1,\ldots,v_n)} = \big((0, \mathsf{W}), \cdot, \cdot\big)$.
By definition, there is no r and P such that $(r, P) <_i (0, \mathsf{W})$, so the statement is trivially true.

Inductive case $t > 0$ **:** We prove the step from $t{-}1$ to t referring to the step

$$\langle\, \mathbf{B}_{t-1}, \mathbf{Q}_{t-1}, \mathbf{S}_{t-1} \,\rangle \;\to_{j:(\textsc{Rule})} \langle\, \mathbf{B}_t, \mathbf{Q}_t, \mathbf{S}_t \,\rangle \tag{1}$$

of R with $\mathbf{S}_{t-1}(j) = \big((r_{t-1}, P_{t-1}), \cdot, \cdot\big)$ and $\mathbf{S}_t(j) = \big((r_t, P_t), \cdot, \cdot\big)$ referring to the program counter of the process j that was touched by the last transition, where the transition rule (Rule) was applied.
By induction,

$$\begin{aligned}&\text{for all } i \in \mathbb{P} \text{ and for all valid program counters } (r, P) \text{ with } (0, \mathsf{W}) \le_i (r, P) <_i (r_{t-1}, P_{t-1})\\ &\text{there is } 0 \le u < t{-}1 \text{ with } \mathbf{S}_u(i) = \big((r, P), \cdot, \cdot\big).\end{aligned} \tag{2}$$

If $i \neq j$, then the desired statement is immediately true by the induction hypothesis, because the transition from $t{-}1$ to t only touches the program counter of process j.
If $i = j$, then we proceed by exhaustive analysis of the cases of (Rule).

case (crash) This rule does not apply since $\mathbf{S}_t(j) \neq \bot$.
cases (Qpp-*) and (Rbc-deliver): With these rules, we get $(r_{t-1}, P_{t-1}) = (r_t, P_t)$, thus the statement holds immediately by the induction hypothesis.

cases (while) and (phs-*) In all these cases, we get $(r_{t-1}, P_{t-1}) <_j (r_t, P_t)$. Moreover, process i proceeds to the *immediate successor* in the program counter ordering, i.e., there is no $(\underline{r}, \underline{P})$ such that $(r_{t-1}, P_{t-1}) <_j (\underline{r}, \underline{P}) <_j (r_t, P_t)$. To conclude, we only require in addition to the induction hypothesis (2) that also for (r_{t-1}, P_{t-1}) there exists a $u < t$ such that $\mathbf{S}_u(i) = \big((r_{t-1}, P_{t-1}), \cdot, \cdot\big)$, which is trivially satisfied with equation (1) by $u = t-1$.

□

The following result states that in a consensus run messages cannot be skipped. More precisely: a process i that has reached some round r, as witnessed by its program counter in the process state, must necessarily have sent 1st- and 3rd-phase messages in all smaller rounds, plus 2nd-phase messages in the rounds smaller than r where i was coordinator.

Lemma 5.3.2 (Messages cannot be skipped) *Let* $\big(\langle\, \mathbf{B}_x, \mathbf{Q}_x, \mathbf{S}_x \,\rangle\big)_T^{(v_1,\ldots,v_n)}$ *be a run. Let* $\mathbf{S}_z(i) = ((r, P), \cdot, \cdot)$*, for some* $z \in T$*. Let* $\hat{r} \in \mathbb{N}$ *with* $\hat{r} \leq r$*.*

1. *If* $(\hat{r}, \mathsf{P1}) <_i (r, P)$*, then* $\mathbf{Q}_z.\mathsf{P1}^{\hat{r}}_{i \to \mathrm{crd}(\hat{r})} \neq \bot$*.*
2. *If* $(\hat{r}, \mathsf{P2}) <_i (r, P)$ *and* $i = \mathrm{crd}(\hat{r})$*, then* $\mathbf{Q}_z.\mathsf{P2}^{\hat{r}}_{\mathrm{crd}(\hat{r}) \to k} \neq \bot$ *for all* $k \in \mathbb{P}$*.*
3. *If* $(\hat{r}, \mathsf{P3}) <_i (r, P)$*, then* $\mathbf{Q}_z.\mathsf{P3}^{\hat{r}}_{i \to \mathrm{crd}(\hat{r})} \neq \bot$*.*

In the following lemma we prove that the stamp contained in the current belief of a process state is always smaller than or equal to the current round number.

Lemma 5.3.3 (Stamp consistency in process states) *Let* $\big(\langle\, \mathbf{B}_x, \mathbf{Q}_x, \mathbf{S}_x \,\rangle\big)_T^{(v_1,\ldots,v_n)}$ *be a run. Let* $t \in T$ *and* $i \in \mathbb{P}$ *with* $\mathbf{S}_t(i) = \big((r_t, P_t), (\cdot, s_t), \cdot\big)$*.*

1. *If* $P_t \in \{\mathsf{P4}, \mathsf{W}\}$ *then* $s_t \leq r_t$*.*
2. *If* $P_t \in \{\mathsf{P1}, \mathsf{P2}, \mathsf{P3}\}$ *then* $s_t < r_t$*.*
3. *If* $z \in T$*, with* $t \leq z$ *and* $\mathbf{S}_z(i) = \big((r_z, P_z), (\cdot, s_z), \cdot\big)$*, then* $s_t \leq s_z$*.*

The previous consistency lemma for process state information directly carries over to messages.

Lemma 5.3.4 (Stamp consistency in process proposal) *Let* $\big(\langle\, \mathbf{B}_x, \mathbf{Q}_x, \mathbf{S}_x \,\rangle\big)_T^{(v_1,\ldots,v_n)}$ *be a run. Let* $\mathbf{Q}_z.\mathsf{P1}^{r}_{i \to \mathrm{crd}(r)} = ((\cdot, s), \cdot)$ *for some* $z \in T$*, round* $r \in \mathbb{N}$*, stamp* $s \in \mathbb{N}$ *and process* $i \in \mathbb{P}$*. Then:*

1. $s < r$
2. *for all* r' *with* $0 < r' < r$*, we have* $\mathbf{Q}_z.\mathsf{P1}^{r'}_{i \to \mathrm{crd}(r')} = ((\cdot, s'), \cdot)$*, for some* $s' \in \mathbb{N}$ *with* $s' \leq s$*.*

Proof.

1. 1st-phase messages can only be created by applying rule (PHS-1), where the belief component of the current state is copied into the new 1st-phase message. As rule (PHS-1) can only be applied in phase P1, by Lemma 5.3.3(2) it follows that $s < r$.
2. As $\mathbf{Q}_z.\mathsf{P1}^{r}_{i \to \mathrm{crd}(r)} = ((\cdot, s), \cdot)$, by Lemma 5.2.2(1) there is $u \in T$ with $u \leq z$, for which

$$\langle\, \mathbf{B}_{u-1}, \mathbf{Q}_{u-1}, \mathbf{S}_{u-1} \,\rangle \;\rightarrow_{i:(\text{PHS-1})}\; \langle\, \mathbf{B}_u, \mathbf{Q}_u, \mathbf{S}_u \,\rangle$$

   where $\mathbf{S}_{u-1}(i) = \big((r, \mathsf{P1}), (\cdot, s), \cdot\big)$ and $\mathbf{S}_u(i) = \big((r, P), (\cdot, s), \cdot\big)$, with $P = \mathsf{P2}$ if $i = \mathrm{crd}(r)$ and $P = \mathsf{P3}$ otherwise.
   Let us fix an arbitrary $r' < r$. By Definition 5.1.1, it holds that $(r', \mathsf{P1}) <_i (r, P)$.
   By Lemma 5.3.2(1), we get that $\mathbf{Q}_u.\mathsf{P1}^{r'}_{i \to \mathrm{crd}(r')} \neq \bot$. Let $\mathbf{Q}_u.\mathsf{P1}^{r'}_{i \to \mathrm{crd}(r')} = ((\cdot, s'), \cdot)$ for some $s' \in \mathbb{N}$. Again, by Lemma 5.2.2(1) there is $t \in T$ for which

$$\langle\, \mathbf{B}_{t-1}, \mathbf{Q}_{t-1}, \mathbf{S}_{t-1} \,\rangle \;\rightarrow_{i:(\text{PHS-1})}\; \langle\, \mathbf{B}_t, \mathbf{Q}_t, \mathbf{S}_t \,\rangle$$

where $\mathbf{S}_{t-1}(i)=\big((r',\mathsf{P1}),(\cdot,s'),\cdot\big)$ and $\mathbf{S}_t(i)=\big((r',P),(\cdot,s'),\cdot\big)$, with $P'\in\{\mathsf{P2},\mathsf{P3}\}$. Since $r'<r$, by Definition 5.1.1 we have $(r',P')<_i(r,\mathsf{P1})$; by Definition 5.1.2 it follows that $\mathbf{S}_t(i)<\mathbf{S}_u(i)$. By Lemma 5.2.1 process states are non-decreasing, and hence $t<u$. By Lemma 5.3.3(3), we derive $s'\le s$. By Corollary 5.2.5 and $\mathbf{Q}_u.\mathsf{P1}^{r'}_{i\to\mathrm{crd}(r')}=((\cdot,s'),\cdot)$, we also have $\mathbf{Q}_z.\mathsf{P1}^{r'}_{i\to\mathrm{crd}(r')}=((\cdot,s'),\cdot)$.

□

The next lemma states how the proposal of a process, i.e., its 1st-phase message, correlates with the 3rd-phase message that the process itself emitted in the previous round. In fact, the 3rd-phase message is an acknowledgment for the round proposal: depending on whether the process acknowledges positively or negatively, it adopts the round proposal as its own new belief or it keeps its old one. It is this possibly updated belief that the process then sends as its proposal to the coordinator of the following round.

Lemma 5.3.5 (Proposal creation) *Let* $\big(\langle\,\mathbf{B}_x,\mathbf{Q}_x,\mathbf{S}_x\,\rangle\big)_T^{(v_1,\ldots,v_n)}$ *be a run. Let* $\mathbf{Q}_z.\mathsf{P1}^r_{i\to\mathrm{crd}(r)}=((v,s),\cdot)$ *and* $\mathbf{Q}_z.\mathsf{P1}^{r+1}_{i\to\mathrm{crd}(r+1)}=((v',s'),\cdot)$, *for some* $z\in T$, *round* $r>0$, *values* $v,v'\in\mathbb{V}$, *and stamps* $s,s'\in\mathbb{N}$.

1. *If* $\mathbf{Q}_z.\mathsf{P3}^r_{i\to\mathrm{crd}(r)}=(\mathrm{nack},\cdot)$, *then* $v=v'$ *and* $s=s'$.
2. *If* $\mathbf{Q}_z.\mathsf{P3}^r_{i\to\mathrm{crd}(r)}=(\mathrm{ack},\cdot)$, *then* $v'=\mathrm{val}^r(\mathbf{Q}_z)$ *and* $s<s'=r$.

Proof.

1. Note that, since $\mathbf{Q}_z.\mathsf{P3}^r_{i\to\mathrm{crd}(r)}=(\mathrm{nack},\cdot)$, rule (PHS-3-SUSPECT) tells us that $i\neq\mathrm{crd}(r)$.
   As $\mathbf{Q}_z.\mathsf{P3}^r_{i\to\mathrm{crd}(r)}=(\mathrm{nack},\cdot)$, by Lemma 5.2.2(3a) there is $u\in T$, $u\le z$, for which
   $$\langle\,\mathbf{B}_{u-1},\mathbf{Q}_{u-1},\mathbf{S}_{u-1}\,\rangle\ \to_{i:(\text{PHS-3-SUSPECT})}\ \langle\,\mathbf{B}_u,\mathbf{Q}_u,\mathbf{S}_u\,\rangle$$
   such that
   $$\begin{array}{rcll}\mathbf{Q}_{u-1}.\mathsf{P3}^r_{i\to\mathrm{crd}(r)}&=&\bot&\qquad\mathbf{S}_{u-1}(i)=\big((r,\mathsf{P3}),\hat{b},\cdot\big)\\ \mathbf{Q}_u.\mathsf{P3}^r_{i\to\mathrm{crd}(r)}&=&(\mathrm{nack},\cdot)&\qquad\mathbf{S}_u(i)=\big((r,\mathsf{W}),\hat{b},\cdot\big)\ .\end{array}$$
   As $\mathbf{Q}_z.\mathsf{P1}^r_{i\to\mathrm{crd}(r)}=((v,s),\cdot)$, by Lemma 5.2.2(1) there is $t\in T$, $t\le z$, for which
   $$\langle\,\mathbf{B}_{t-1},\mathbf{Q}_{t-1},\mathbf{S}_{t-1}\,\rangle\ \to_{i:(\text{PHS-1})}\ \langle\,\mathbf{B}_t,\mathbf{Q}_t,\mathbf{S}_t\,\rangle$$
   such that
   $$\begin{array}{rcll}\mathbf{Q}_{t-1}.\mathsf{P1}^r_{i\to\mathrm{crd}(r)}&=&\bot&\qquad\mathbf{S}_{t-1}(i)=\big((r,\mathsf{P1}),(v,s),\cdot\big)\\ \mathbf{Q}_t.\mathsf{P1}^r_{i\to\mathrm{crd}(r)}&=&\big((v,s),\cdot\big)&\qquad\mathbf{S}_t(i)=\big((r,\mathsf{P3}),(v,s),\cdot\big)\ .\end{array}$$
   As $\mathbf{Q}_z.\mathsf{P1}^{r+1}_{i\to\mathrm{crd}(r+1)}=\big((v',s'),\cdot\big)$, by Lemma 5.2.2(1) there is $w\in T$, $w\le z$, for which
   $$\langle\,\mathbf{B}_{w-1},\mathbf{Q}_{w-1},\mathbf{S}_{w-1}\,\rangle\ \to_{i:(\text{PHS-1})}\ \langle\,\mathbf{B}_w,\mathbf{Q}_w,\mathbf{S}_w\,\rangle$$
   such that
   $$\begin{array}{rcll}\mathbf{Q}_{w-1}.\mathsf{P1}^{r+1}_{i\to\mathrm{crd}(r+1)}&=&\bot&\qquad\mathbf{S}_{w-1}(i)=\big((r{+}1,\mathsf{P1}),(v',s'),\cdot\big)\\ \mathbf{Q}_w.\mathsf{P1}^{r+1}_{i\to\mathrm{crd}(r+1)}&=&\big((v',s'),\cdot\big)&\qquad\mathbf{S}_w(i)=\big((r{+}1,P),(v',s'),\cdot\big)\ .\end{array}$$
   where, if $i=\mathrm{crd}(r{+}1)$ then $P=\mathsf{P2}$, otherwise $P=\mathsf{P3}$.
   By Lemma 5.2.1 process states are non-decreasing. As $\mathbf{S}_t(i)<\mathbf{S}_u(i)<\mathbf{S}_w(i)$, it follows that $t<u<w$ and the three transitions above must be ordered in time as follows:
   $$\begin{array}{lllll}\langle\,\mathbf{B}_0,\mathbf{Q}_0,\mathbf{S}_0^{(v_1,\ldots,v_n)}\,\rangle&\to^*&\langle\,\mathbf{B}_{t-1},\mathbf{Q}_{t-1},\mathbf{S}_{t-1}\,\rangle&\to_{i:(\text{PHS-1})}&\langle\,\mathbf{B}_t,\mathbf{Q}_t,\mathbf{S}_t\,\rangle\\ &\to^*&\langle\,\mathbf{B}_{u-1},\mathbf{Q}_{u-1},\mathbf{S}_{u-1}\,\rangle&\to_{i:(\text{PHS-3-SUSPECT})}&\langle\,\mathbf{B}_u,\mathbf{Q}_u,\mathbf{S}_u\,\rangle\\ &\to^*&\langle\,\mathbf{B}_{w-1},\mathbf{Q}_{w-1},\mathbf{S}_{w-1}\,\rangle&\to_{i:(\text{PHS-1})}&\langle\,\mathbf{B}_w,\mathbf{Q}_w,\mathbf{S}_w\,\rangle\\ &\to^*&\langle\,\mathbf{B}_z,\mathbf{Q}_z,\mathbf{S}_z\,\rangle\ .&&\end{array}$$

Since $i \neq \mathrm{crd}(r)$, and since $\mathbf{S}_t(i)$ and $\mathbf{S}_{u-1}(i)$ have the same program counter $(r, \mathsf{P3})$, the only rules involving process i that can have been applied in the derivation sequence

$$\langle\, \mathbf{B}_t, \mathbf{Q}_t, \mathbf{S}_t \,\rangle \;\rightarrow^* \langle\, \mathbf{B}_{u-1}, \mathbf{Q}_{u-1}, \mathbf{S}_{u-1} \,\rangle$$

are (RBC-DELIVER), (QPP_SND), and (QPP_DELIVER). None of these rules change the current belief of process i. As a consequence, $\hat{b} = (v, s)$.
A similar reasoning applies to the derivation sequence

$$\langle\, \mathbf{B}_u, \mathbf{Q}_u, \mathbf{S}_u \,\rangle \;\rightarrow^* \langle\, \mathbf{B}_{w-1}, \mathbf{Q}_{w-1}, \mathbf{S}_{w-1} \,\rangle$$

where $(r{+}1, \mathsf{P1})$ of $\mathbf{S}_{w-1}$ is the immediate successor to (r, W) of $\mathbf{S}_u$. Thus, in the displayed sequence, rule (WHILE) *must* have been applied, while otherwise the only rules involving process i that *can* have been applied are (RBC-DELIVER), (QPP_SND), and (QPP_DELIVER). None of these rules change the current belief of process i. As a consequence, $\hat{b}{=}(v', s')$, but this implies also $(v', s'){=}(v, s)$. So, $\mathbf{Q}_w.\mathsf{P1}^{r+1}_{i\rightarrow\mathrm{crd}(r+1)}{=}((v, s), \cdot)$. By Corollary 5.2.5, we get $\mathbf{Q}_z.\mathsf{P1}^{r+1}_{i\rightarrow\mathrm{crd}(r+1)}{=}((v, s), \cdot)$, as required.

2. As $\mathbf{Q}_z.\mathsf{P3}^{r}_{i\rightarrow\mathrm{crd}(r)} = (\mathrm{ack}, \cdot)$, by Lemma 5.2.2(3b) there is $t \in T$, $t \leq z$, for which

$$\langle\, \mathbf{B}_{t-1}, \mathbf{Q}_{t-1}, \mathbf{S}_{t-1} \,\rangle \;\rightarrow_{i:(\text{PHS-3-TRUST})} \langle\, \mathbf{B}_t, \mathbf{Q}_t, \mathbf{S}_t \,\rangle$$

such that

$$\begin{array}{rclcrcl} \mathbf{Q}_{t-1}.\mathsf{P3}^{r}_{i\rightarrow\mathrm{crd}(r)} & = & \bot & & \mathbf{S}_{t-1}(i) & = & \big((r, \mathsf{P3}), \cdot, \cdot\big) \\ \mathbf{Q}_{t}.\mathsf{P3}^{r}_{i\rightarrow\mathrm{crd}(r)} & = & (\mathrm{ack}, \cdot) & & \mathbf{S}_{t}(i) & = & \big((r, P), (\mathrm{val}^r(\mathbf{Q}_{t-1}), r), \cdot\big) \end{array}$$

where, if $i = \mathrm{crd}(r)$ then $P = \mathsf{P4}$, otherwise $P = \mathsf{W}$.
As $\mathbf{Q}_z.\mathsf{P1}^{r+1}_{i\rightarrow\mathrm{crd}(r+1)} = \big((v', s'), \cdot\big)$, by Lemma 5.2.2(1) there is $u \in T$, $u \leq z$, for which

$$\langle\, \mathbf{B}_{u-1}, \mathbf{Q}_{u-1}, \mathbf{S}_{u-1} \,\rangle \;\rightarrow_{i:(\text{PHS-1})} \langle\, \mathbf{B}_u, \mathbf{Q}_u, \mathbf{S}_u \,\rangle$$

such that,

$$\begin{array}{rclcrcl} \mathbf{Q}_{u-1}.\mathsf{P1}^{r+1}_{i\rightarrow\mathrm{crd}(r+1)} & = & \bot & & \mathbf{S}_{u-1}(i) & = & \big((r{+}1, \mathsf{P1}), (v', s'), \cdot\big) \\ \mathbf{Q}_{u}.\mathsf{P1}^{r+1}_{i\rightarrow\mathrm{crd}(r+1)} & = & \big((v', s'), \cdot\big) & & \mathbf{S}_{u}(i) & = & \big((r{+}1, P'), (v', s'), \cdot\big) \end{array}$$

where, if $i = \mathrm{crd}(r{+}1)$ then $P' = \mathsf{P2}$, otherwise $P' = \mathsf{P3}$.
By Lemma 5.2.1 process states are non-decreasing. As $\mathbf{S}_t(i) < \mathbf{S}_u(i)$, it follows that $t < u$, and the two transitions are ordered in time as follows:

$$\begin{array}{lcllll} \langle\, \mathbf{B}_0, \mathbf{Q}_0, \mathbf{S}_0^{(v_1,\ldots,v_n)} \,\rangle & \rightarrow^* & \langle\, \mathbf{B}_{t-1}, \mathbf{Q}_{t-1}, \mathbf{S}_{t-1} \,\rangle & \rightarrow_{i:(\text{PHS-3-TRUST})} & \langle\, \mathbf{B}_t, \mathbf{Q}_t, \mathbf{S}_t \,\rangle \\ & \rightarrow^* & \langle\, \mathbf{B}_{u-1}, \mathbf{Q}_{u-1}, \mathbf{S}_{u-1} \,\rangle & \rightarrow_{i:(\text{PHS-1})} & \langle\, \mathbf{B}_u, \mathbf{Q}_u, \mathbf{S}_u \,\rangle \\ & \rightarrow^* & \langle\, \mathbf{B}_z, \mathbf{Q}_z, \mathbf{S}_z \,\rangle & & \end{array}$$

By Lemma 5.2.1 process states are non-decreasing. As a consequence, the only rules involving process i that can have been applied in the derivation sequence $\langle\, \mathbf{B}_t, \mathbf{Q}_t, \mathbf{S}_t \,\rangle \;\rightarrow^* \langle\, \mathbf{B}_{u-1}, \mathbf{Q}_{u-1}, \mathbf{S}_{u-1} \,\rangle$ and that can possibly have affected its state are: (PHS-4-FAIL), (PHS-4-SUCCESS), (WHILE) and (RBC-DELIVER). However, none of these rules affects the belief component of the state, and hence $(v', s'){=}(\mathrm{val}^r(\mathbf{Q}_{t-1}), r)$. By Corollary 5.2.6 we have $\mathrm{val}^r(\mathbf{Q}_{t-1}){=}\mathrm{val}^r(\mathbf{Q}_z)$. As a consequence, $v' = \mathrm{val}^r(\mathbf{Q}_z)$ and $s' = r$. Finally, since $\mathbf{Q}_z.\mathsf{P1}^{r}_{i\rightarrow\mathrm{crd}(r)} = ((v, s), \cdot)$, by Lemma 5.3.4(1) it follows that $s < r$.

□

We need a last technical lemma in order to show how process proposals are generated.

Lemma 5.3.6 *Let* $\big(\langle\, \mathbf{B}_x, \mathbf{Q}_x, \mathbf{S}_x \,\rangle\big)_T^{(v_1,\ldots,v_n)}$ *be a run. Let* $z \in T$, $r, r' \in \mathbb{N}$ *with* $0 < r' < r$, *and* $i \in \mathbb{P}$ *such that* $\mathbf{Q}_z.\mathsf{P1}^{r'}_{i\to\mathrm{crd}(r')}=((v', s'), \cdot)$ *and* $\mathbf{Q}_z.\mathsf{P1}^{r}_{i\to\mathrm{crd}(r)}=((v, s), \cdot)$.

1. *If* $s' = s$ *then* $v' = v$ *and for all* r'', $r' \le r'' < r$, *it holds that* $\mathbf{Q}_z.\mathsf{P3}^{r''}_{i\to\mathrm{crd}(r'')}=(\mathrm{nack}, \cdot)$.
2. *If* $s' < s$ *then there is a round* $\hat{r}$, $r'\le\hat{r}<r$, *such that* $\mathbf{Q}_z.\mathsf{P3}^{\hat{r}}_{i\to\mathrm{crd}(\hat{r})}=(\mathrm{ack}, \cdot)$.

The following lemma states that the proposal (v, s) proposed by some process i in a certain round r originates precisely from round s.

Lemma 5.3.7 (Proposal origin) *Let* $\big(\langle\, \mathbf{B}_x, \mathbf{Q}_x, \mathbf{S}_x \,\rangle\big)_T^{(v_1,\ldots,v_n)}$ *be a run and* $z \in T$.
If $\mathbf{Q}_z.\mathsf{P1}^{r}_{i\to\mathrm{crd}(r)} = ((v, s), \cdot)$, *with* $r > 1$ *and* $s > 0$, *then* $\mathbf{Q}_z.\mathsf{P3}^{s}_{i\to\mathrm{crd}(s)} = (\mathrm{ack}, \cdot)$ *and* $v = \mathrm{val}^s(\mathbf{Q}_z)$.

Proof. By $\mathbf{Q}_z.\mathsf{P1}^{r}_{i\to\mathrm{crd}(r)} = ((v, s), \cdot)$ and Lemma 5.3.4(1) it follows that $s < r$. By Lemma 5.3.4(2) and $s < r$, we derive $\mathbf{Q}_z.\mathsf{P1}^{s}_{i\to\mathrm{crd}(s)} \neq \bot$ and $\mathbf{Q}_z.\mathsf{P1}^{s+1}_{i\to\mathrm{crd}(s+1)} \neq \bot$ (the case $s{+}1 = r$ is trivial). Let $\mathbf{Q}_z.\mathsf{P1}^{s}_{i\to\mathrm{crd}(s)} = ((v_s, r_s), \cdot)$, by Lemma 5.3.4(1) we have $r_s < s$.

Now, suppose by contradiction that $\mathbf{Q}_z.\mathsf{P3}^{s}_{i\to\mathrm{crd}(s)} = (\mathrm{nack}, \cdot)$. Then, with Lemma 5.3.5(1), also $\mathbf{Q}_z.\mathsf{P1}^{s+1}_{i\to\mathrm{crd}(s+1)} = ((v_s, r_s), \cdot)$. As $r_s < s$, and by (matching $s{+}1/r_s$ with r'/s' of) Lemma 5.3.6(2), there is some $\hat{r}$ with $s{+}1 \le \hat{r} < r$ (and hence $s < \hat{r}$) such that $\mathbf{Q}_z.\mathsf{P3}^{\hat{r}}_{i\to\mathrm{crd}(\hat{r})} = (\mathrm{ack}, \cdot)$. By an application of Lemma 5.3.5(2), we get $\mathbf{Q}_z.\mathsf{P1}^{\hat{r}+1}_{i\to\mathrm{crd}(\hat{r}+1)} = ((\mathrm{val}^{\hat{r}}(\mathbf{Q}_z), \hat{r}), \cdot)$. As $\mathbf{Q}_z.\mathsf{P1}^{r}_{i\to\mathrm{crd}(r)} = ((v, s), \cdot)$, by an application Lemma 5.3.4(2) we get the contradiction $\hat{r} \le s$. As a consequence, it must be $\mathbf{Q}_z.\mathsf{P3}^{s}_{i\to\mathrm{crd}(s)} = (\mathrm{ack}, \cdot)$.

By Lemma 5.3.5(2) and $\mathbf{Q}_z.\mathsf{P3}^{s}_{i\to\mathrm{crd}(s)} = (\mathrm{ack}, \cdot)$, we get $\mathbf{Q}_z.\mathsf{P1}^{s+1}_{i\to\mathrm{crd}(s+1)} = ((\mathrm{val}^s(\mathbf{Q}_z), s), \cdot)$. This together with Lemma 5.3.6(1) allows us to derive $v = \mathrm{val}^s(\mathbf{Q}_z)$.

□

The next lemma states that if the coordinator broadcasts a value in a certain round, then it must have previously—in the very same round—received a majority of (positive) acknowledgments on its round proposal.

Lemma 5.3.8 (Coordinator's RBC-broadcast) *Let* $\big(\langle\, \mathbf{B}_x, \mathbf{Q}_x, \mathbf{S}_x \,\rangle\big)_T^{(v_1,\ldots,v_n)}$ *be a run, and* $z \in T$. *If* $\mathbf{B}_z(r)\neq\bot$ *then* $|\,\mathrm{ack}(\mathrm{received}(\mathbf{Q}_z).\mathsf{P3}^r)| \ge \lceil (n{+}1)/2 \rceil$.

Proof. If $\mathbf{B}(r)\neq\bot$ by Lemma 5.2.2(4) there is $t \in T$, for which

$$\langle\, \mathbf{B}_{t-1}, \mathbf{Q}_{t-1}, \mathbf{S}_{t-1} \,\rangle \;\rightarrow_{\mathrm{crd}(r):(\textsc{phs-4-success})}\; \langle\, \mathbf{B}_t, \mathbf{Q}_t, \mathbf{S}_t \,\rangle$$

such that, for some $v \in \mathbb{V}$

$$\begin{array}{rclcrcl} \mathbf{B}_{t-1}(r) & = & \bot & & \mathbf{S}_{t-1}(\mathrm{crd}(r)) & = & \big((r, \mathsf{P4}), (v, \cdot), \cdot\big) \\ \mathbf{B}_{t}(r) & = & (v, \mathbb{P}) & & \mathbf{S}_{t}(\mathrm{crd}(r)) & = & \big((r, \mathsf{W}), (v, \cdot), \cdot\big)\ . \end{array}$$

The premises of rule (PHS-4-SUCCESS) require $|\,\mathrm{ack}(\mathrm{received}(\mathbf{Q}_{t-1}).\mathsf{P3}^r)| \ge \lceil (n{+}1)/2 \rceil$. Again, by Lemma 5.2.4, message histories are non-decreasing and hence $\mathbf{Q}_{t-1} \le \mathbf{Q}_t \le \mathbf{Q}_z$. As a consequence,

$$|\,\mathrm{ack}(\mathrm{received}(\mathbf{Q}_z).\mathsf{P3}^r)| \ge |\,\mathrm{ack}(\mathrm{received}(\mathbf{Q}_t).\mathsf{P3}^r)| \ge |\,\mathrm{ack}(\mathrm{received}(\mathbf{Q}_{t-1}).\mathsf{P3}^r)| \ge \lceil (n{+}1)/2 \rceil\ .$$

□

6 Consensus properties

In this section, we prove the three main Consensus properties: *Validity*, *Agreement*, and *Termination*.

Apart from Validity (Section 6.1), in our opinion, it is not obvious how to turn the intuitive, but rather informal, correctness arguments of Section 3.2 into rigorous proofs. Most of the arguments given there use the concept of round, which is orthogonal to the concept of time in a run. With our proof of Agreement (Section 6.2) and its supporting set of lemmas, we demonstrate how to carefully recover the events belonging to some round from the details of a given run. The high-level proof structure is very similar to Chandra and Toueg's proof in [CT96]. For Termination (Section 6.3), however, we needed to come up with different proof ideas and structures, mainly because our underlying notion of run is different from the one in [CT96].

6.1 Validity

Intuitively, in a generic run, whenever a value appears as an estimate, a proposal, or a decision — i.e., within one of the components $\mathbf{B}$, $\mathbf{Q}$ or $\mathbf{S}$— then this value is one of the initially proposed values. In other words, values are never invented. The Validity property of the Consensus algorithm coincides with the last item of the following theorem.

Theorem 6.1.1 (Validity) *Let* $\big(\langle\, \mathbf{B}_x, \mathbf{Q}_x, \mathbf{S}_x \,\rangle\big)_T^{(v_1,\ldots,v_n)}$ *be a run. Then, for all* $t \in T$*, the conjunction of the following properties holds:*

1. *If* $\mathbf{B}_t(r) = (v, \cdot)$*, for some* $0 < r$*, then there is* $k \in \mathbb{P}$ *with* $v = v_k$*.*
2. *If* $\mathbf{Q}_t.\mathsf{P1}^r_{i\to\mathrm{crd}(r)} = ((v,\cdot),\cdot)$*, for some* $i \in \mathbb{P}$ *and* $0 < r$*, then there is* $k \in \mathbb{P}$ *with* $v = v_k$*.*
3. *If* $\mathbf{Q}_t.\mathsf{P2}^r_{\mathrm{crd}(r)\to i} = ((v,\cdot),\cdot)$*, for some* $i \in \mathbb{P}$ *and* $0 < r$*, then there is* $k \in \mathbb{P}$ *with* $v = v_k$*.*
4. *If* $\mathbf{S}_t(i) = \big(\cdot,(v,\cdot),\cdot\big)$*, for some* $i \in \mathbb{P}$*, then there is* $k \in \mathbb{P}$ *with* $v = v_k$*.*
5. *If* $\mathbf{S}_t(i) = \big(\cdot,\cdot,(v,\cdot)\big)$*, for some* $i \in \mathbb{P}$*, then there is* $k \in \mathbb{P}$ *with* $v = v_k$*.*

Proof. By induction on the length t of the (prefix of a) run $\langle\, \mathbf{B}_0, \mathbf{Q}_0, \mathbf{S}_0^{(v_1,\ldots,v_n)} \,\rangle \;\to^t\; \langle\, \mathbf{B}_t, \mathbf{Q}_t, \mathbf{S}_t \,\rangle$.

Base case $t = 0$ **:** Immediate since $\mathbf{B}_0 = \emptyset$, $\mathbf{Q}_0 = \emptyset$ and $\mathbf{S}(i) = \mathbf{S}_0^{(v_1,\ldots,v_n)} = \big((0,\mathsf{W}),(v_i,0),\bot\big)$ for $i \in \mathbb{P}$.

Inductive case $t > 0$ **:** We prove the step from $t-1$ to t (if $t \in T$) referring to the step

$$\langle\, \mathbf{B}_{t-1}, \mathbf{Q}_{t-1}, \mathbf{S}_{t-1} \,\rangle \;\to_{j:(\textsc{Rule})}\; \langle\, \mathbf{B}_t, \mathbf{Q}_t, \mathbf{S}_t \,\rangle \tag{3}$$

By exhaustive analysis, we check for every rule (Rule) that each of the five conditions is satisfied in configuration $\langle\, \mathbf{B}_t, \mathbf{Q}_t, \mathbf{S}_t \,\rangle$. All cases are similar: in every possible step, values that appear in the conclusions of the applied rule are just copied via the premises referring to the former configuration, such that the inductive hypothesis applies and proves the required origin in the initial configuration.

case (Rule) = (phs-2) for some $0 < r$**:**

1. $\mathbf{B}_t = \mathbf{B}_{t-1}$, so the inductive hypothesis is already sufficient.
2. $\mathbf{Q}_t.\mathsf{P1}^r = \mathbf{Q}_{t-1}.\mathsf{P1}^r$ for all $0 < r$, so the inductive hypothesis is already sufficient.
3. $\mathbf{Q}_t = \mathbf{Q}_{t-1}\{\, \mathsf{P2}^r_{\mathrm{crd}(r)\to j} \mapsto (\mathrm{best}^r(\mathbf{Q}_{t-1}), \mathrm{outgoing}) \,\}$, so in addition to the inductive hypothesis on $\mathbf{Q}_{t-1}$, we only need to show that $\mathrm{best}^r(\mathbf{Q}_{t-1}) = v_k$ for some $k \in \mathbb{P}$. This is guaranteed by the projective definition of $\mathrm{best}^r(\cdot)$ together with the inductive hypothesis on $\mathbf{Q}_{t-1}$.
4. Since $\mathbf{S}_{t-1}(j) = ((r,\mathsf{P2}),b,d)$ and $\mathbf{S}_t(j) = ((r,\mathsf{P3}),b,d)$ mention the same value in b, and since $\mathbf{S}_t(i) = \mathbf{S}_{t-1}(i)$ for all $i \neq j$, the inductive hypothesis is already sufficient.
5. Completely analogous to the previous case, except arguing with d instead of b.

case (Rule) $\neq$ **(phs-2):** Analogous to the previous, just that there is not even an indirection referring to an externally defined function like $\mathrm{best}^r(\cdot)$.

□

6.2 Agreement

In the Introduction we said that a value gets *locked* as soon as enough processes have, in the same round, positively acknowledged the estimate proposed by the coordinator of that round (the round proposal). In our terminology, a value v is *locked* for round r (in a message history $\mathbf{Q}$) if enough positive 3rd-phase messages are defined for round r; the transmission state of these ack-messages is not important, be it outgoing, transit, or received.

Definition 6.2.1 *A value v is called* locked *for round r in a* QPP*-message history $\mathbf{Q}$, written $\mathbf{Q} \overset{r}{\Rightarrow} v$, if $v = \mathrm{val}^r(\mathbf{Q})$ and $|\,\mathrm{ack}(\mathbf{Q}.\mathsf{P3}^r)| \geq \lceil (n{+}1)/2 \rceil$.*

Notice the convenience of having at hand the history abstraction to access the messages that were sent in the past, without having to look at the run leading to the current state. This is independent of possible crashes of the senders of the messages.

We now show that whenever a RBC-broadcast occurs, it is for a locked value.

Lemma 6.2.2 (Locking) *Let $\big(\langle\, \mathbf{B}_x, \mathbf{Q}_x, \mathbf{S}_x \,\rangle\big)_T^{(v_1,\ldots,v_n)}$ be a run where $\mathbf{B}_z(r) = (v, \cdot)$ for some $z \in T$, $0 < r \in \mathbb{N}$, $v \in \mathbb{V}$. Then $\mathbf{Q}_z \overset{r}{\Rightarrow} v$.*

Proof. Since $\mathbf{B}_0(r) = \bot$ and $\mathbf{B}_z(r) \neq \bot$, by Lemma 5.2.2(4), there exists $t \in T$, with $t \leq z$, for which

$$\langle\, \mathbf{B}_{t-1}, \mathbf{Q}_{t-1}, \mathbf{S}_{t-1} \,\rangle \;\rightarrow_{\mathrm{crd}(r):(\text{PHS-4-SUCCESS})}\; \langle\, \mathbf{B}_t, \mathbf{Q}_t, \mathbf{S}_t \,\rangle$$

such that, for some $\hat{v} \in \mathbb{V}$ and $\hat{r} \in \mathbb{N}$,

$$\begin{array}{rclcrcl} \mathbf{B}_{t-1}(r) & = & \bot & \qquad & \mathbf{S}_{t-1}(\mathrm{crd}(r)) & = & \big((r,\mathsf{P4}),(\hat{v},\hat{r}),\cdot\big) \\ \mathbf{B}_t(r) & = & (\hat{v},\mathbb{P}) & & \mathbf{S}_t(\mathrm{crd}(r)) & = & \big((r,\mathsf{W}),(\hat{v},\hat{r}),\cdot\big)\ . \end{array}$$

As $(r,\mathsf{P3}) <_{\mathrm{crd}(r)} (r,\mathsf{P4})$, by Lemma 5.3.2(3), we have $\mathbf{Q}_t.\mathsf{P3}^r_{\mathrm{crd}(r)\to\mathrm{crd}(r)} \neq \bot$. As the coordinator cannot suspect itself, it must be $\mathbf{Q}_t.\mathsf{P3}^r_{\mathrm{crd}(r)\to\mathrm{crd}(r)} = (\mathrm{ack},\cdot)$. Thus, by an application of Lemma 5.2.2(3b), there exists $u \in T$, $u < t$ (notice that it cannot be $u = t$), for which

$$\langle\, \mathbf{B}_{u-1}, \mathbf{Q}_{u-1}, \mathbf{S}_{u-1} \,\rangle \;\rightarrow_{\mathrm{crd}(r):(\text{PHS-3-TRUST})}\; \langle\, \mathbf{B}_u, \mathbf{Q}_u, \mathbf{S}_u \,\rangle$$

such that $\mathbf{S}_{u-1}(\mathrm{crd}(r)) = \big((r,\mathsf{P3}),\cdot,\cdot\big)$ and $\mathbf{S}_u(\mathrm{crd}(r)) = \big((r,\mathsf{P4}),(\mathrm{val}^r(\mathbf{Q}_{u-1}),r),\cdot\big)$.

Since $\mathbf{S}_u(\mathrm{crd}(r))$ and $\mathbf{S}_{t-1}(\mathrm{crd}(r))$ have the same program counter $(r,\mathsf{P4})$, the only rules involving $\mathrm{crd}(r)$ that can have been applied in $\langle\, \mathbf{B}_u, \mathbf{Q}_u, \mathbf{S}_u \,\rangle \rightarrow^* \langle\, \mathbf{B}_{t-1}, \mathbf{Q}_{t-1}, \mathbf{S}_{t-1} \,\rangle$ are (QPP_SND), (QPP_DELIVER) and (RBC-DELIVER). However, rules (QPP_SND) and (QPP_DELIVER) do not affect the state of process $\mathrm{crd}(r)$, while (RBC-DELIVER) affects only the decision component while keeping the belief component unchanged. As a consequence, $\hat{v} = \mathrm{val}^r(\mathbf{Q}_{u-1})$ and $\hat{r} = r$.

As $\mathbf{B}_z(r) = (v,\cdot)$, $\mathbf{B}_t(r) = (\hat{v},\cdot)$ and $t \leq z$, by Corollary 5.2.5(2), we then have $v = \hat{v} = \mathrm{val}^r(\mathbf{Q}_{u-1})$. By Corollary 5.2.6, we obtain

$$v = \mathrm{val}^r(\mathbf{Q}_z)\ .$$

Moreover, since $\mathbf{B}_z(r) \neq \bot$, by Lemma 5.3.8 we have

$$|\,\mathrm{ack}(\mathbf{Q}_z.\mathsf{P3}^r)| \geq |\,\mathrm{ack}(\mathrm{received}(\mathbf{Q}_z).\mathsf{P3}^r)| \geq \lceil (n{+}1)/2 \rceil\ .$$

From these two results, by applying Definition 6.2.1, we derive $\mathbf{Q}_z \overset{r}{\Rightarrow} v$.

□

Now, we can move to look at agreement properties. The key idea is to prove that lockings in two different rounds must be for the very same value.

Proposition 6.2.3 (Locking agreement) *Let* $\big(\langle\, \mathbf{B}_x, \mathbf{Q}_x, \mathbf{S}_x \,\rangle\big)_T^{(v_1,\ldots,v_n)}$ *be a run. If there are a state* $z \in T$*, round numbers* $r_1, r_2 \in \mathbb{N}$ *and values* $v_1, v_2 \in \mathbb{V}$ *such that* $\mathbf{Q}_z \overset{r_1}{\hookrightarrow} v_1$ *and* $\mathbf{Q}_z \overset{r_2}{\hookrightarrow} v_2$*, then* $v_1 = v_2$*.*

Proof. Let us assume $r_1 \leq r_2$ (the proof for the opposite case is analogous). From Definition 6.2.1, we have $v_1 = \mathrm{val}^{r_1}(\mathbf{Q}_z)$ and $v_2 = \mathrm{val}^{r_2}(\mathbf{Q}_z)$. We will prove that $\mathrm{val}^{r_1}(\mathbf{Q}_z) = \mathrm{val}^{r_2}(\mathbf{Q}_z)$ by strong induction on r_2 starting from r_1.

Base case If $r_1 = r_2$ then the result is trivially true.
Inductive case Let $r_2 > r_1$.

By inductive hypothesis we assume that for all $r \in \mathbb{N}$, $r_1 \leq r \leq r_2{-}1$,

$$\mathrm{val}^{r}(\mathbf{Q}_z) = \mathrm{val}^{r_1}(\mathbf{Q}_z) = v_1.$$

By definition of $\mathrm{val}^{r_2}(\mathbf{Q}_z)$ (Definition 4.2.2), we have $\mathbf{Q}_z.\mathsf{P2}^{r_2}_{\mathrm{crd}(r_2)\to i} = ((\mathrm{val}^{r_2}(\mathbf{Q}_z), \cdot), \cdot)$, for all $i \in \mathbb{P}$. By Lemma 5.2.2(2), there is $t \in T$, $t \leq z$, for which

$$\langle\, \mathbf{B}_{t-1}, \mathbf{Q}_{t-1}, \mathbf{S}_{t-1} \,\rangle \;\rightarrow_{\mathrm{crd}(r_2):(\textsc{phs}\text{-}2)}\; \langle\, \mathbf{B}_t, \mathbf{Q}_t, \mathbf{S}_t \,\rangle$$

such that, for some $b \in \mathbb{V} \times \mathbb{N}$ and for all $i \in \mathbb{P}$

$$\begin{array}{rclcrcl} \mathbf{Q}_{t-1}.\mathsf{P2}^{r_2}_{\mathrm{crd}(r_2)\to i} & = & \bot & & \mathbf{S}_{t-1}(\mathrm{crd}(r_2)) & = & \big((r_2, \mathsf{P2}), b, \cdot\big) \\ \mathbf{Q}_{t}.\mathsf{P2}^{r_2}_{\mathrm{crd}(r_2)\to i} & = & (\mathrm{best}^{r_2}(\mathbf{Q}_{t-1}), \mathrm{outgoing}) & & \mathbf{S}_{t}(\mathrm{crd}(r_2)) & = & \big((r_2, \mathsf{P3}), b, \cdot\big)\ . \end{array}$$

From Table 3, we see that one of the premises of rule (PHS-2) is $|\,\mathrm{received}(\mathbf{Q}_{t-1}).\mathsf{P1}^{r_2}| \geq \lceil (n{+}1)/2 \rceil$.
Let $C_{r_2} = \{\, j \mid \mathbf{Q}_z.\mathsf{P1}^{r_2}_{j\to\mathrm{crd}(r_2)} = (\cdot, \mathrm{received}) \,\}$, then by Corollary 5.2.5(1) on the monotonicity on message histories we have $|\,C_{r_2}\,| \geq |\,\mathrm{received}(\mathbf{Q}_{t-1}).\mathsf{P1}^{r_2}\,| \geq \lceil (n{+}1)/2 \rceil$.
Let $L_{r_1} = \{\, j \mid \mathbf{Q}_z.\mathsf{P3}^{r_1}_{j\to\mathrm{crd}(r_1)} = (\mathrm{ack}, \cdot) \,\}$. As $\mathbf{Q}_z \overset{r_1}{\hookrightarrow} v_1$, by Definition 6.2.1 we have $|\,\mathrm{ack}(\mathbf{Q}_z.\mathsf{P3}^{r_1})| \geq \lceil (n{+}1)/2 \rceil$ and hence $|\,L_{r_1}\,| \geq \lceil (n{+}1)/2 \rceil$.
Since the cardinality of both C_{r_2} and L_{r_1} is greater than $\lceil (n{+}1)/2 \rceil$, and since $C_{r_2}, L_{r_1} \subseteq \{1, \ldots, n\}$, it follows that $L_{r_1} \cap C_{r_2} \neq \emptyset$. Let us take a process $k \in L_{r_1} \cap C_{r_2}$.
As $k \in C_{r_2}$, we have $\mathbf{Q}_z.\mathsf{P1}^{r_2}_{k\to\mathrm{crd}(r_2)} \neq \bot$. As $r_2 \geq r_1{+}1$, by Lemma 5.3.4(2) also $\mathbf{Q}_z.\mathsf{P1}^{r_1+1}_{k\to\mathrm{crd}(r_1+1)} \neq \bot$.
Since $k \in L_{r_1}$, we have $\mathbf{Q}_z.\mathsf{P3}^{r_1}_{k\to\mathrm{crd}(r_1)} = (\mathrm{ack}, \cdot)$. By Lemma 5.3.5(2)

$$\mathbf{Q}_z.\mathsf{P1}^{r_1+1}_{k\to\mathrm{crd}(r_1+1)} = \big(\big(\,\mathrm{val}^{r_1}(\mathbf{Q}_z), r_1\big), \cdot\big)\ .$$

Since $k \in C_{r_2}$, we have

$$\mathbf{Q}_z.\mathsf{P1}^{r_2}_{k\to\mathrm{crd}(r_2)} = \big(\big(\cdot, \tilde{r}\big), \mathrm{received}\,\big)$$

for some $\tilde{r} \in \mathbb{N}$. As $r{+}1 \leq r_2$, by Lemma 5.3.4(2) we have $r_1 \leq \tilde{r}$.
Let us now consider the process $i \in C_{r_2}$ that has proposed the best value in round r_2, according to Definition 4.2.1. Then by Definition 4.2.2 we have

$$\mathbf{Q}_z.\mathsf{P1}^{r_2}_{i\to\mathrm{crd}(r_2)} = \big(\big(\,\mathrm{val}^{r_2}(\mathbf{Q}_z), \hat{r}\big), \mathrm{received}\,\big)$$

for some $\hat{r} \in \mathbb{N}$. By applying Lemma 5.3.4(1), we get $\hat{r} < r_2$. As i has proposed in round r_2 the best value, with the best timestamp $\hat{r}$, it follows that $\tilde{r} \leq \hat{r}$.
Putting it altogether, we get $r_1 \leq \tilde{r} \leq \hat{r} \leq r_2{-}1$. By the inductive hypothesis, $\mathrm{val}^{\hat{r}}(\mathbf{Q}_z) = \mathrm{val}^{r_1}(\mathbf{Q}_z)$. From $\mathbf{Q}_z.\mathsf{P1}^{r_2}_{i\to\mathrm{crd}(r_2)} = \big(\big(\,\mathrm{val}^{r_2}(\mathbf{Q}_z), \hat{r}\big), \mathrm{received}\,\big)$ and Lemma 5.3.7, we get that $\mathrm{val}^{\hat{r}}(\mathbf{Q}_z) = \mathrm{val}^{r_2}(\mathbf{Q}_z)$. Thus, $\mathrm{val}^{r_1}(\mathbf{Q}_z) = \mathrm{val}^{r_2}(\mathbf{Q}_z)$, as required.

□

Now, everything is in place to prove the property of Agreement: Whenever two processes i and j have decided, as expressed within the respective state components, they have done so for the same value.

Theorem 6.2.4 (Agreement) *Let* $\big(\langle \mathbf{B}_x, \mathbf{Q}_x, \mathbf{S}_x \rangle\big)_T^{(v_1,\ldots,v_n)}$ *be a run. If there are a time* $z \in T$*, processes* $i, j \in \mathbb{P}$*, and values* $v_1, v_2 \in \mathbb{V}$ *such that* $\mathbf{S}_z(i) = (\cdot, \cdot, (v_1, \cdot))$ *and* $\mathbf{S}_z(j) = (\cdot, \cdot, (v_2, \cdot))$*, then* $v_1 = v_2$*.*

Proof. In the initial state $\mathbf{S}_0$ the decision component of both process i and process j is undefined (see Section 4.1), while in state $\mathbf{S}_z$ it is defined. By inspection of the rules of Table 3, the decision element of a process state can be changed only once per run, by applying rule (RBC-DELIVER). In particular, it can change from $\perp$ to something of the form $(\hat{v}, \hat{r})$ executing rule (RBC-DELIVER) for the first time. When executing rule (RBC-DELIVER) the following times, the decision value is left unchanged. Therefore, once initialized, the decision value of a process state stays the same until the end of the run. As a consequence, there are $t \in T$ and $u \in T$, with $t \leq z$ and $u \leq z$, such that

$$\langle \mathbf{B}_{t-1}, \mathbf{Q}_{t-1}, \mathbf{S}_{t-1} \rangle \rightarrow_{i:(\text{RBC-DELIVER})} \langle \mathbf{B}_t, \mathbf{Q}_t, \mathbf{S}_t \rangle$$

with $\mathbf{S}_{t-1}(i) = (\cdot, \cdot, \perp)$, $\mathbf{S}_t(i) = (\cdot, \cdot, (v_1, r_1))$, $\mathbf{B}_{t-1}(r_1) = (v_1, \cdot)$ for some $r_1 > 0$, and

$$\langle \mathbf{B}_{u-1}, \mathbf{Q}_{u-1}, \mathbf{S}_{u-1} \rangle \rightarrow_{j:(\text{RBC-DELIVER})} \langle \mathbf{B}_u, \mathbf{Q}_u, \mathbf{S}_u \rangle$$

with $\mathbf{S}_{u-1}(j) = (\cdot, \cdot, \perp)$, $\mathbf{S}_u(i) = (\cdot, \cdot, (v_2, r_2))$, $\mathbf{B}_{u-1}(r_2) = (v_2, \cdot)$ for some $r_2 > 0$.

As $u, t \leq z$, by Corollary 5.2.5(2), $\mathbf{B}_z(r_1) = (v_1, \cdot)$ and $\mathbf{B}_z(r_2) = (v_2, \cdot)$.

By Lemma 6.2.2, it follows that $\mathbf{Q}_z \overset{r_1}{\Rightarrow} v_1$ and $\mathbf{Q}_z \overset{r_2}{\Rightarrow} v_2$.

By Proposition 6.2.3, we conclude $v_1 = v_2$.

□

Recall that the *Agreement* property states that two correct processes cannot decide different values. As mentioned in §2.4, there is a stronger property, called *Uniform Agreement*, requiring that any two processes (no matter whether correct or faulty) cannot decide different values. Since in the proof of Theorem 6.2.4 the correctness of processes is not required, the Chandra and Toueg's algorithm actually satisfies the property of *Uniform Agreement* [CT96].

6.3 Termination

The layout and conduct of the proof of Termination is strongly influenced by the fundamental definition of runs and their required properties. A standard approach in the field of Distributed Algorithms is that of considering system runs of infinite length. This is also the path followed by Chandra and Toueg [CT96]: they work under the important fairness assumption that "every correct process takes an infinite number of steps". On the other hand, when working with transition systems, as we do in this paper, it makes sense to admit complete runs of finite length (cf. Section 4). We find it more adequate and intuitive to capture the termination of a single Distributed Consensus by means of a run that takes only finitely many steps. In fact, for complete finite runs, a particular fairness property holds automatically: since no transition is enabled in the final configuration of a complete finite run, any previously (permanently) enabled transition must have been executed in this run.

To compare the different layouts of Chandra and Toueg's proof of Termination and ours, we first briefly sketch how the former works. The critical part of [CT96, Lemma 6.2.2 (Termination)] is to capture the notion of *blocking of processes* when waiting for messages or circumstances that might never arrive. This argument is formulated as "*no correct process remains blocked forever at one of the* **wait** *statements*". The associated reasoning (by contradiction) starts with the identification, in a given run, of a process that is blocked in some program counter; should there be several processes that are blocked in the run, the one with the smallest program counter is chosen. As we understand it, a process is blocked in a given run if there exists a configuration after which the process stays forever in the same program counter. Thus, blocking can only occur if (A) no transition is ever enabled, or –note that we consider infinite runs, aka: linear-time behaviors– if (B) an enabled transition is never executed. Due to the above-mentioned fairness assumption on "correct process progress" in infinite runs and the fact that processes are

essentially single-threaded, every enabled transition would eventually be executed, so case B is immediately excluded. Then, under carefully chosen assumptions of the contradiction setup one can show that transitions would always eventually become enabled, so also case A is excluded. For this reasoning, the main ingredients are (1) the fairness assumptions on QPP- and RBC-delivery, (2) the FD-properties of $\diamond\mathcal{S}$ and (3) the assumption on a majority of correct processes.

The decisive role of the definition of infinite runs, and its accompanying fairness assumption, in Chandra and Toueg's proof is obvious. Not surprisingly, since our definition of run is different in that it admits finite runs and comes without explicit fairness assumptions, also our Termination proof is different. Moreover, our proof layout is conveniently structured in that it allows us to separate concerns. First, we prove as an independent result that admissible runs are always finite. Then, the bare analysis of the last configuration of an admissible run allows us to reason about the blocking of processes in a very simple way; no fairness assumptions need to be considered, because no transitions are enabled anyway.

We start proving that all admissible runs of Consensus (Definition 4.5.5) are finite. The proof relies on the next lemma showing that in a hypothetic infinite run of Consensus every round $r \in \mathbb{N}$ is reached by at least one process. Intuitively, this holds since the (finite) set of processes cannot stay within their respective rounds for an infinite number of computation steps. Unless all processes decide or crash, at least one of them must proceed to higher rounds. The reader may observe some similarity to the "correct process progress" fairness assumption in infinite runs of Chandra and Toueg.

Lemma 6.3.1 (Infinity) *Let* $\big(\langle\, \mathbf{B}_t, \mathbf{Q}_t, \mathbf{S}_t \,\rangle\big)_{\mathbb{N}}^{(v_1,\ldots,v_n)}$ *be an infinite run of Consensus. Then, for all rounds* $r > 0$*, there are* $i \in \mathbb{P}$ *and* $t \in \mathbb{N}$ *such that* $\mathbf{S}_t(i) = \big((r, \mathsf{P3}), \cdot, \cdot\big)$.

Proof. By contradiction. Assume there is $r > 0$ such that, for all $i \in \mathbb{P}$ and $t \in \mathbb{N}$, it holds that $\mathbf{S}_t(i) \neq \big((r, \mathsf{P3}), \cdot, \cdot\big)$. By applying Lemma 5.3.1 we have:

$$\text{for all } i \in \mathbb{P} \text{ and } t \in \mathbb{N}, \text{ if } \mathbf{S}_t(i) = \big((r_t, P_t), \cdot, \cdot\big), \text{ then } (r_t, P_t) <_i (r, \mathsf{P3}). \tag{4}$$

Note that, for every process i, the number of program counters (r_t, P_t) smaller than $(r, \mathsf{P3})$ is finite, and since there are only five phases, the number of the rounds that can be reached by any process i is finite too.

In the following, we show that the number of reductions that a process may perform at a certain program counter (r_t, P_t) is finite. For this, we compute an upper bound on the maximum number of possible derivations that a process may perform at program counter (r_t, P_t). We proceed by case analysis on the transition rules for an arbitrary $i \in \mathbb{P}$:

1. The rules (WHILE), (PHS-1), (PHS-2), (PHS-3-TRUST), (PHS-3-SUSPECT), (PHS-4-FAIL), (PHS-4-SUCCESS), and (CRASH) cause the change of the program counter. Thus, by Lemma 5.2.1, each of them can only be performed once for each program counter.
2. All remaining rules do not change the program counter, so they are critical for the desired bound.
   (a) The rule (RBC-DELIVER) can be performed for each occurrence of some $\mathbf{B}(\hat{r}) = (v, D)$ such that D contains i. However, every execution of (RBC-DELIVER) removes i from the respective D, which corresponds to the property of RBC-Integrity of Definition 4.5.4, such that for each $\hat{r}$, process i can execute (RBC-DELIVER) only once. How many possible $\hat{r}$ are there? Entries $\mathbf{B}(\hat{r})$ can only be produced by the rule (PHS-4-SUCCESS), once per round number $\hat{r}$ that has been reached by the corresponding coordinator. Since we have shown above with (4) that only finitely many round numbers are reached, there are only finitely many possible applications of (RBC-DELIVER) for process i.
   (b) The rule (QPP_SND) can be performed at most $r_t{*}(n{+}2)$ times; where $n{+}2$ is the maximum number of point-to-point messages that a process may produce in a round. We multiply this number for r_t assuming the worst case that none of the messages produced in the previous rounds has yet left the site.

(c) The rule (QPP_DELIVER) can be performed at most $r_t*(2*n+1)$ times. In fact, the message addressed to a certain process in a given round are at most $2*n+1$, when the process is coordinator of the round. We multiply this number for r_t supposing that none of the messages sent in the previous rounds has yet been received by the site.

Summing up, since there are only finitely many possible program counters (r_t, P_t) smaller than $(r, \mathsf{P3})$, and since each process can only perform a finite number of steps while staying at such a counter, the whole run can only contain a finite number of computation steps. This concludes the contradiction. □

Now, everything is in place to prove that all the admissible runs of Consensus are finite. This is done by contradiction, assuming that there is an infinite run of Consensus. In this case, the presence of FD Ω and the constraint that only a minority of processes can crash ensure that, at a certain round, some coordinator broadcasts its decision. The properties of Reliable Broadcast are then used to show that the run should be finite, thus obtaining a contradiction.

Theorem 6.3.2 (Finiteness of Admissible Runs) *All admissible runs of Consensus are finite.*

Proof. We proceed by contradiction. Assume there is an *infinite* admissible run of Consensus

$$R = \big(\langle\, \mathbf{B}_x, \mathbf{Q}_x, \mathbf{S}_x \,\rangle\big)_{\mathbb{N}}^{(v_1,\ldots,v_n)} .$$

As an auxiliary definition, let $\text{max\_rnd}(\mathbf{Q}_x) := \max\{\, r \mid (r,\cdot,\cdot,\cdot,\cdot,\cdot) \in \mathbf{Q}_x \,\}$ denote the greatest round number for which, by time x, any QPP-message has occurred.

As incorrect processes are doomed to crash, there is a time $t_{\text{crashed}} \in \mathbb{N}$ such that for all $u \geq t_{\text{crashed}}$ and $j \in \mathbb{P} \setminus \mathsf{Correct}(R)$ it holds that $\mathbf{S}_u(j) = \bot$. As R is admissible, the Ω-property (Definition 4.5.3) ensures that there is a time $t_{\text{trusted}} \in \mathbb{N}$ after which some process $i \in \mathsf{Correct}(R)$ is never again suspected. Let $t = \max\{t_{\text{trusted}}, t_{\text{crashed}}\}$. Let r be the smallest round such that $r > \text{max\_rnd}(\mathbf{Q}_t)$ and $i = \text{crd}(r)$. By this construction, as process i is no longer suspected after time t, process i is not suspected in R in any round greater or equal than r. Also, if $(r,\cdot,\cdot,\cdot,\cdot,\cdot) \in \mathbf{Q}_u$, at some time u, then $t < u$.

In the following, we first derive that, in R, process i passes through round r and that it executes rule (PHS-4-SUCCESS) while doing so. Using this knowledge, in a second step, we use the RBC-properties of the admissible run R to derive a contradiction.

First, let us move to round $r+n+1$. As R is infinite, by Lemma 6.3.1 there must exist a time $z \in \mathbb{N}$, such that for some process $k \in \mathbb{P}$ we have $\mathbf{S}_z(k) = \big((r+n+1, \mathsf{P3}), \cdot, \cdot\big)$.

Now, we move backwards to round $r+n$. As $(r+n, \mathsf{P3}) <_k (r+n+1, \mathsf{P3})$, by Lemma 5.3.2(3) and $i = \text{crd}(r) = \text{crd}(r+n)$, we get $\mathbf{Q}_z.\mathsf{P3}_{k\to i}^{r+n} \neq \bot$. By Lemma 5.2.2(3), in round $r+n$, process k must have executed either rule (PHS-3-SUSPECT) or rule (PHS-3-TRUST). However, as process i is not suspected in any round beyond r according to Definition 4.5.3, it is rule (PHS-3-TRUST) that process k must have executed in round $r+n$. Thus, with Lemma 5.2.2(3b) we have $\mathbf{Q}_z.\mathsf{P3}_{k\to i}^{r+n} = (\text{ack}, \cdot)$. Moreover, the premises of (PHS-3-TRUST) require the presence of the 2nd-phase message sent by the coordinator i to process k. Formally, by Corollary 5.2.5(1), $\mathbf{Q}_z.\mathsf{P2}_{i\to k}^{r+n} \neq \bot$. By Lemma 5.2.2(2), this implies that in round $r+n$ process i executed rule (PHS-2) in state $\mathbf{S}_w(i) = \big((r+n, \mathsf{P2}), \cdot, \cdot\big)$ for some $w \in \mathbb{N}$. (Intuitively, this means that the coordinator i must itself have reached round $r+n$.)

Now, we move back to round r. As $i = \text{crd}(r)$ and $(r, \mathsf{P4}) <_i (r, \mathsf{W}) <_i (r+n, \mathsf{P2})$, by Lemma 5.3.1 and Lemma 5.2.1 there is a time $u < w$ such that $\mathbf{S}_{u-1}(i) = \big((r, \mathsf{P4}), \cdot, \cdot\big)$ and $\mathbf{S}_u(i) = \big((r, \mathsf{W}), \cdot, \cdot\big)$. By inspection of the transition rules of Tables 2 and 3, the state $\mathbf{S}_u(i)$ can only be reached by executing either rule (PHS-4-SUCCESS) or rule (PHS-4-FAIL). The premises of both rules require that in round r process i must have received a majority of 3rd-phase messages from the other processes. Formally, $|\,\text{received}(\mathbf{Q}_{u-1}).\mathsf{P3}^r| \geq \lceil (n+1)/2 \rceil$. As process i is not suspected in round r, it follows that i must have received in round r only *positive* 3rd-phase messages. Formally, $|\,\text{ack}(\text{received}(\mathbf{Q}_{u-1}).\mathsf{P3}^r)| \geq \lceil (n+1)/2 \rceil$. As a consequence, process i in round r must have reached the state $\mathbf{S}_u(i) = \big((r, \mathsf{W}), \cdot, \cdot\big)$ by firing rule (PHS-4-SUCCESS).

By the definition of rule (PHS-4-SUCCESS) and Lemma 5.2.3(4), we have $\mathbf{B}_{u-1}(r) = \bot$ and $\mathbf{B}_u(r) = (v, \mathbb{P})$ for some $v \in \mathbb{V}$ and $r \in \mathbb{N}$. This means that in round r process i has RBC-broadcast the value v. As R is admissible (Definition 4.5.5), the *Validity* RBC-*property* (Definition 4.5.4(1)) guarantees that, in R, process i eventually RBC-delivers the value v by executing rule (RBC-DELIVER). Likewise, the *Agreement* RBC-*property* (Definition 4.5.4(2)) guarantees that, in R, all processes in $\mathsf{Correct}(R)$ eventually RBC-deliver v and decide by executing rule (RBC-DELIVER). Formally, with Definition 4.5.4(2) and rule (RBC-DELIVER), there is a time $t_{\text{delivered}}$ such that for each $j \in \mathsf{Correct}(R)$ there exists a value $d_j \neq \bot$ such that $\mathbf{S}_{t_{\text{delivered}}}(j) = \big((\cdot,\cdot),\cdot,d_j\big)$. Notice that rule (RBC-DELIVER) is the only one that acts on the decision component of process states; moreover, once this component is defined its value cannot change afterwards.

Now, we are ready to derive the contradiction. Observe that, by construction (recall that R is infinite), we have $t_{\text{crashed}} \leq t < u < t_{\text{delivered}}$. Let us consider some round $\hat{r} > 1 + \max\_\text{rnd}(\mathbf{Q}_{t_{\text{delivered}}})$. As R is infinite, by Lemma 6.3.1, there are $k \in \mathbb{P}$ and $z \in \mathbb{N}$ such that $\mathbf{S}_z(k) = \big((\hat{r}, \mathsf{P3}),\cdot,\cdot\big)$. Since $\hat{r}$ is only reached well beyond time t_{crashed}, we have that $k \in \mathsf{Correct}(R)$. Since program counters cannot be skipped, by Lemma 5.3.1, there must be a time where k passed through both previous subsequent counters $(\hat{r}-1, \mathsf{W}) <_k (\hat{r}, \mathsf{P1})$. The only way to do this is by executing rule (WHILE) at round $\hat{r}-1$, but this is not possible since according to our construction $d_k \neq \bot$ must have been already defined at round $\hat{r} - 1 > \max\_\text{rnd}(\mathbf{Q}_{t_{\text{delivered}}})$. Contradiction. □

In the remainder of the paper, we prove that all correct processes of an admissible Consensus run eventually decide in that run. Some terminology will be helpful to discuss this theorem.

Definition 6.3.3 (Process termination and deadlock) *Let*

$$\langle\, \mathbf{B}_0, \mathbf{Q}_0, \mathbf{S}_0^{(v_1,\ldots,v_n)} \,\rangle \;\rightarrow^* \langle\, \mathbf{B}, \mathbf{Q}, \mathbf{S} \,\rangle \not\rightarrow$$

be a finite run of Consensus. With respect to the final state of this run, a process $i \in \mathbb{P}$ is called

1. crashed *if* $\mathbf{S}(i) = \bot$*;*
2. deadlocked *if* $\mathbf{S}(i) = \big(\cdot,\cdot,\bot\big)$*;*
3. terminated *if* $\mathbf{S}(i) = \big(\cdot,\cdot,d\big)$*, with* $d \neq \bot$*.*

Note that both *deadlocked* and *terminated* processes represent *blocked* behaviors, but due to the condition $d \neq \bot$ the latter is considered successful, while the former is not. Since the three cases of the Definition 6.3.3 cover all possibilities for the possible states of $\mathbf{S}(i)$, it is obvious that in the final state, all processes are either crashed, deadlocked, or terminated.

Since we know by Theorem 6.3.2 that admissible runs are finite, we may conveniently focus our analysis of process termination on the last configuration of a given admissible run. To support this kind of reasoning, the following lemma describes the precise conditions under which processes can be deadlocked in such a final configuration, depending on their role in the algorithm at this last moment.

Lemma 6.3.4 (Process deadlock) *Let* $R = \big(\langle\, \mathbf{B}_t, \mathbf{Q}_t, \mathbf{S}_t \,\rangle\big)_{[0,\ldots,z]}^{(v_1,\ldots,v_n)}$ *be an admissible run of Consensus. Let $i \in \mathbb{P}$ be deadlocked, that is* $\mathbf{S}_z(i) = ((r, P),\cdot,\bot)$*, for some r and P.*

1. *If* $i \neq \text{crd}(r)$ *then* $P = \mathsf{P3}$*,* $\mathbf{S}_z(\text{crd}(r)) \neq \bot$*, and* $\mathbf{Q}_z.\mathsf{P2}^r_{\text{crd}(r)\to i} = \bot$*;*
2. *If* $i = \text{crd}(r)$ *then*
   (a) *either* $P = \mathsf{P2}$ *and* $|\,\text{received}(\mathbf{Q}_z).\mathsf{P1}^r\,| < \lceil (n+1)/2 \rceil$
   (b) *or* $P = \mathsf{P4}$ *and* $|\,\text{received}(\mathbf{Q}_z).\mathsf{P3}^r\,| < \lceil (n+1)/2 \rceil$.

Proof. As R is complete, but finite, we have that $\langle\, \mathbf{B}_z, \mathbf{Q}_z, \mathbf{S}_z \,\rangle \not\rightarrow$.

1. Let $i \neq \text{crd}(r)$. We proceed by case analysis on the possible phases $P \in \{\, \mathsf{P1}, \mathsf{P3}, \mathsf{W} \,\}$ of non-coordinating participants.

If $P = \mathsf{P1}$ then rule (PHS-1) could be applied without further condition. As $\langle\, \mathbf{B}_z, \mathbf{Q}_z, \mathbf{S}_z \,\rangle \not\rightarrow$, it follows that $P \neq \mathsf{P1}$.

If $P = \mathsf{W}$ then rule (WHILE) could be applied since $\mathbf{S}_z(i)) = \big((r, P), \cdot, \bot\big)$. As $\langle\, \mathbf{B}_z, \mathbf{Q}_z, \mathbf{S}_z \,\rangle \not\rightarrow$, it follows that $P \neq \mathsf{W}$.

If $P = \mathsf{P3}$ then two rules might apply: (PHS-3-TRUST) and (PHS-3-SUSPECT). As $\langle\, \mathbf{B}_z, \mathbf{Q}_z, \mathbf{S}_z \,\rangle \not\rightarrow$ both rules must not meet the conditions to fire. More precisely, since rule (PHS-3-TRUST) cannot apply, then it must be $\mathbf{Q}_z.\mathsf{P2}^r_{\mathrm{crd}(r)\rightarrow i} \neq (\cdot, \mathrm{received})$. By definition of QPP-reliability (Definition 4.5.2) this implies $\mathbf{Q}_z.\mathsf{P2}^r_{\mathrm{crd}(r)\rightarrow i} = \bot$. Moreover, since rule (PHS-3-SUSPECT) cannot apply then $\mathrm{crd}(r)$ has not crashed, i.e. $\mathbf{S}_z(\mathrm{crd}(r)) \neq \bot$.

2. Let $i = \mathrm{crd}(r)$. Again, we proceed by case analysis on $P \in \{\, \mathsf{P1}, \mathsf{P2}, \mathsf{P3}, \mathsf{P4}, \mathsf{W} \,\}$. The cases $P = \mathsf{P1}$ and $P = \mathsf{W}$ are identical to the previous case of $i \neq \mathrm{crd}(r)$, since the respective rules are independent of i being $\mathrm{crd}(r)$, or not.

   case P2 The only possibility to prevent an application of rule (PHS-2) is by *not* satisfying $|\,\mathrm{received}(\mathbf{Q}_z).\mathsf{P1}^r\,| \geq \lceil (n{+}1)/2 \rceil$, so we have $|\,\mathrm{received}(\mathbf{Q}_z).\mathsf{P1}^r\,| < \lceil (n{+}1)/2 \rceil$.

   case P3 This case is different from the respective one for $i \neq \mathrm{crd}(r)$ that we have seen above: with $i = \mathrm{crd}(r)$ only the rule (PHS-3-TRUST) *may* be applicable. And we show here that it actually *is*. By Lemma 5.3.2(2), since $(r, \mathsf{P2}) <_i (r, \mathsf{P3})$ and $i = \mathrm{crd}(r)$, we derive $\mathbf{Q}_z.\mathsf{P2}^r_{\mathrm{crd}(r)\rightarrow i} \neq \bot$. As $\langle\, \mathbf{B}_z, \mathbf{Q}_z, \mathbf{S}_z \,\rangle \not\rightarrow$ and $\mathbf{S}_z(i) \neq \bot$ it follows that $i \in \mathsf{Correct}(R)$. By Definition 4.5.2, it follows that $\mathbf{Q}_z.\mathsf{P2}^r_{\mathrm{crd}(r)\rightarrow i} = (\cdot, \mathrm{received})$. Thus, rule (PHS-3-TRUST) applies to the final state $\mathbf{S}_z(i) = ((r, \mathsf{P3}), \cdot, \cdot)$. As $\langle\, \mathbf{B}_z, \mathbf{Q}_z, \mathbf{S}_z \,\rangle \not\rightarrow$, it follows that $P \neq \mathsf{P3}$.

   case P4 Potentially, either of the rules (PHS-4-SUCCESS) or (PHS-4-FAIL) may apply. The only possibility to prevent an application of rule (PHS-4-FAIL) is by *not* satisfying $|\,\mathrm{received}(\mathbf{Q}_z).\mathsf{P3}^r\,| \geq \lceil (n{+}1)/2 \rceil$, thus we have $|\,\mathrm{received}(\mathbf{Q}_z).\mathsf{P3}^r\,| < \lceil (n{+}1)/2 \rceil$. As

   $$|\,\mathrm{ack}(\mathrm{received}(\mathbf{Q}_z).\mathsf{P3}^r)\,| \leq |\,\mathrm{received}(\mathbf{Q}_z).\mathsf{P3}^r\,|$$

   the condition $|\,\mathrm{received}(\mathbf{Q}_z).\mathsf{P3}^r\,| < \lceil (n{+}1)/2 \rceil$ is sufficient to prevent also an application of rule (PHS-4-SUCCESS), which requires $|\,\mathrm{ack}(\mathrm{received}(\mathbf{Q}_z).\mathsf{P3}^r)\,| \geq \lceil (n{+}1)/2 \rceil$.

□

We can now prove the Termination property. Again, we reason by contradiction, assuming that there exists an admissible Consensus run R in which one of the correct processes does not decide. The Agreement RBC-property is then used to derive that, in this case, no correct process decides in R. As no process decides and as admissible runs are finite, it follows that all correct processes should be deadlocked in R. This leads us to the required contradiction.

Theorem 6.3.5 (Termination) *Let* $R = \big(\langle\, \mathbf{B}_x, \mathbf{Q}_x, \mathbf{S}_x \,\rangle\big)_T^{(v_1,\ldots,v_n)}$ *be an admissible run of Consensus.*
If $i \in \mathsf{Correct}(R)$*, then there is a time* $t \in T$ *and a value* d *such that* $\mathbf{S}_t(i) = (\cdot, \cdot, d)$*, with* $d \neq \bot$*.*

Proof. By Theorem 6.3.2, we have $R = \big(\langle\, \mathbf{B}_x, \mathbf{Q}_x, \mathbf{S}_x \,\rangle\big)_{[0,\ldots,z]}^{(v_1,\ldots,v_n)}$, for some $z \in \mathbb{N}$, with $\langle\, \mathbf{B}_z, \mathbf{Q}_z, \mathbf{S}_z \,\rangle \not\rightarrow$.

By contradiction, assume there is a process $i \in \mathsf{Correct}(R)$ that does not decide in R. Formally, we assume that, for all $t \in T$, we have $\mathbf{S}_t(i) = \big(\cdot, \cdot, \bot\big)$. In particular, we have $\mathbf{S}_z(i) = \big(\cdot, \cdot, \bot\big)$.

By inspection of the transition rules in Tables 2 and 3, rule (RBC-DELIVER) is the only one acting on (i.e., defining) the decision component of process states; moreover, once this component is defined, its value remains unchanged. As a consequence, in R, process i cannot have performed rule (RBC-DELIVER). As R is admissible, the *Agreement* RBC*-property* (Definition 4.5.4(2)) guarantees that no process in $\mathsf{Correct}(R)$ RBC-delivered, and hence decided, in R. As a consequence, for all $k \in \mathsf{Correct}(R)$, we have $\mathbf{S}_z(k) = ((r_k, P_k), \cdot, \bot)$, i.e., all processes in $\mathsf{Correct}(R)$ must be deadlocked in R.

Let $\underline{k} \in \mathsf{Correct}(R)$ be the process with the smallest program counter in $\mathsf{Correct}(R)$, formally $(r_{\underline{k}}, P_{\underline{k}}) \leq (r_k, P_k)$, for all $k \in \mathsf{Correct}(R)$. Note that, since $\mathsf{Correct}(R) \neq \emptyset$, such a $\underline{k}$ always exists, although it is not uniquely defined; we simply choose one among possibly several candidates.

By Lemma 6.3.4, there are three possible cases for process $\underline{k}$ being deadlocked at time z:

case $\underline{k} \neq \mathrm{crd}(r_{\underline{k}})$ **and** $P_{\underline{k}} = \mathbf{P3}$. By Lemma 6.3.4, we also have that $\mathbf{S}_z(\mathrm{crd}(r_{\underline{k}})) \neq \bot$ and, for all $k \in \mathsf{Correct}(R)$, $\mathbf{Q}_z.\mathsf{P2}^{r_{\underline{k}}}_{\mathrm{crd}(r_{\underline{k}})\to k} = \bot$. As $\mathbf{S}_z(\mathrm{crd}(r_{\underline{k}})) \neq \bot$, it follows that $\mathrm{crd}(r_{\underline{k}}) \in \mathsf{Correct}(R)$. As $\mathbf{Q}_z.\mathsf{P2}^{r_{\underline{k}}}_{\mathrm{crd}(r_{\underline{k}})\to k} = \bot$, for all $k \in \mathsf{Correct}(R)$, by Lemma 5.3.2(2), the program counter of process $\mathrm{crd}(r_{\underline{k}})$ in $\mathbf{S}_z$ must be smaller or equal than $(r_{\underline{k}}, \mathsf{P2})$. However, this is in contradiction with the fact that $\underline{k}$ has the smallest program counter (which is equal to $(r_{\underline{k}}, \mathsf{P3})$) among the processes in $\mathsf{Correct}(R)$.

case $\underline{k} = \mathrm{crd}(r_{\underline{k}})$ **and** $P_{\underline{k}} = \mathbf{P2}$. By Lemma 6.3.4, we also have $|\,\mathrm{received}(\mathbf{Q}_z).\mathsf{P1}^{r_{\underline{k}}}\,| < \lceil (n{+}1)/2 \rceil$. Moreover, due to the minimality of $\underline{k}$, we have $(r_{\underline{k}}, \mathsf{P1}) < (r_{\underline{k}}, \mathsf{P2}) = (r_{\underline{k}}, P_{\underline{k}}) \leq (r_k, P_k)$, for all $k \in \mathsf{Correct}(R)$. By Lemma 5.3.2(1), it follows that $\mathbf{Q}_z.\mathsf{P1}^{r_{\underline{k}}}_{k\to\mathrm{crd}(r_{\underline{k}})} \neq \bot$, for all $k \in \mathsf{Correct}(R)$. As R is admissible and $\underline{k} \in \mathsf{Correct}(R)$, and by QPP-reliability (Definition 4.5.2), it follows that $\mathbf{Q}_z.\mathsf{P1}^{r_{\underline{k}}}_{k\to\mathrm{crd}(r_{\underline{k}})} = (\cdot, \mathrm{received})$, for all $k \in \mathsf{Correct}(R)$. Thus, $|\,\mathrm{received}(\mathbf{Q}_z).\mathsf{P1}^{r_{\underline{k}}}\,| = |\,\mathsf{Correct}(R)\,| \geq \lceil (n{+}1)/2 \rceil$, in contradiction to what we derived at the beginning of this case.

case $\underline{k} = \mathrm{crd}(r_{\underline{k}})$ **and** $P_{\underline{k}} = \mathbf{P4}$. By Lemma 6.3.4, we also have $|\,\mathrm{received}(\mathbf{Q}_z).\mathsf{P3}^{r}\,| < \lceil (n{+}1)/2 \rceil$. Moreover, due to minimality of $\underline{k}$, we have $(r_{\underline{k}}, \mathsf{P3}) < (r_{\underline{k}}, \mathsf{P4}) = (r_{\underline{k}}, P_{\underline{k}}) \leq (r_k, P_k)$, for all $k \in \mathsf{Correct}(R)$. By Lemma 5.3.2(3), it follows that $\mathbf{Q}_z.\mathsf{P3}^{r_{\underline{k}}}_{k\to\mathrm{crd}(r_{\underline{k}})} \neq \bot$, for all $k \in \mathsf{Correct}(R)$. As R is admissible and $\underline{k} \in \mathsf{Correct}(R)$, and by QPP-reliability (Definition 4.5.2), it follows that $\mathbf{Q}_z.\mathsf{P3}^{r_{\underline{k}}}_{k\to\mathrm{crd}(r_{\underline{k}})} = (\cdot, \mathrm{received})$ for all $k \in \mathsf{Correct}(R)$. Thus, $|\,\mathrm{received}(\mathbf{Q}_z).\mathsf{P3}^{r_{\underline{k}}}\,| = |\,\mathsf{Correct}(R)\,| \geq \lceil (n{+}1)/2 \rceil$, in contradiction to what we derived at the beginning of this case.

These three cases allow us to conclude that there is no process i with $\mathbf{S}_z(i) = (\cdot, \cdot, \bot)$, i.e., deadlocked in the last configuration of R. Thus, for all $k \in \mathsf{Correct}(R)$, it must be $\mathbf{S}_z(k) = (\cdot, \cdot, d_k)$, with $d_k \neq \bot$. □

7 Conclusion

We have presented a global transition-based model to formalize Chandra and Toueg's algorithm for Distributed Consensus among asynchronous processes in the presence of quasi-reliable channels, Ω failure detectors, and under the hypothesis that only a minority of processes may crash. We have used our model to formally prove the correctness of the algorithm.

The essential novelty in the article is the global transition-based representation of the reachable configurations of a Consensus system that formally captures the round-based abstraction. This allowed us to bridge the gap between the description of the algorithm (local to a single process) and the round-based reasoning that encompasses all processes and enables comprehensible structured proofs.

The use of message histories provided us with a tractable way to perform a formal analysis of the past of system runs. Our correctness proof relies on a number of crucial properties of system runs and configurations. Such properties are proved by induction either on round numbers or on the length of system runs, and particular attention is given in avoiding mixing up the two proof techniques.

In the field of Distributed Algorithms, many proofs rely on round-based reasoning, an approach that, to our knowledge, has never been formally justified remaining somewhat vague. Thus, we expect that our contribution will not only be valuable for this particular verification exercise, but also generally improve the understanding of distributed algorithms in asynchronous systems.

Related work. The interest in formalizing distributed algorithms has been quickly growing in the past twenty years. Many researchers have proposed formal methods to describe and prove distributed algorithms, and many have tried to verify particular algorithms.

In 1987, the need for a more rigorous way to describe and prove distributed algorithms led Lynch and Tuttle to introduce I/O Automata [LT87]: a simple but powerful model of computation that allows the construction of modular, hierarchical correctness proofs via the composition of automata with only three possible kinds of actions: input, output and internal. Many extended or specialized variants of I/O Automata have then appeared, for example Probabilistic I/O Automata [Seg95, SL95, WSS97] for the verification of certain randomized distributed algorithms.

In his thesis [Jon87], Jonsson presented a method for the specification and verification of distributed systems that handled both safety and liveness properties. The method was compositional and specification was given as labeled transition systems with fairness properties. Since specifications could be given at various levels of abstraction, Jonsson's method was particularly suitable in a development process where a detailed implementation was developed from an abstract specification through a sequence of refinement steps.

In 1988, Chandy and Misra proposed a new theory—a computational model and a proof theory—called Unbounded Nondeterministic Iterative Transformations, better known with the name of UNITY [CM88]. The computational model consisted of unbounded nondeterministic iterative transformations of the program state, while transformations of the program state were represented by multiple-assignment statements. The theory could be used for the development of programs for a variety of architectures and applications, since it attempted to decouple the programmer's thinking about a program and its implementation on an architecture, i.e., to separate the concerns of *what* from those of *when*, *where* or *how*.

In 1991, Groote and Ponse [GP91], proposed a proof theory that formalized the interaction between processes and data for their μ-CRL (a language that offered a framework for the integrated specification of processes and data). Such proof theory served two purposes: it made it possible to express how the correctness of a protocol depends on data and it enabled precise proofs of correctness of concurrent systems and programs.

Some time later, Lamport came up with a specification language intended for the high-level specification of reactive, distributed and asynchronous systems: TLA+. Combining the linear-time temporal logic TLA and classical set-theory, TLA+ provided an expressive specification formalism and supported assertional verification [Lam02], as represented by invariants and simulation techniques.

Groote and Springintveld [GS01] presented a strategy, expressed in the form of definitions and theorems and described in the settings of μ-CRL, for finding correctness proofs for communication systems. The method extended an algebra of communicating processes with a formal treatment of the interaction between data and processes, it incorporated assertional methods within an algebraic framework, and centered around the idea that the state spaces of distributed systems are structured as a number of so-called *cones* with *focus points*. As a result, this proof strategy reduced a large part of algebraic protocol verification to the checking of a number of elementary facts concerning the data parameters that occur in implementation and specification. One important example of application of this method was given by Fredlund, Groote and Korver [FGK97] who presented a formal description and formal proofs of correctness of the Dolev, Klawe and Rodeh's leader election algorithm: a round-based algorithm that has been designed for a network with a unidirectional ring topology. In each round, every active process exchanges messages only with its neighbors and the number of electable processes decreases until the last process left declares itself the leader.

Since many of the strategies for the verification of concurrent systems are based on simulation techniques, Lynch and Vaandrager [LV95] gave a unified and comprehensive presentation of simulation proof methods for untimed automata, including refinements, forward and backward simulations and combinations thereof, including history and prophecy relations. The paper presented the mutual relationships between all these kinds of simulations, plus soundness and completeness results.

Many of these theories on representation, formalization and verification have also been applied to "real-life" algorithms. For example, Felty and Stomp [FS96] presented the formalization

and verification of cache coherence, a portion of the IEEE Scalable Coherent Interface used for specifying communication between multiprocessors in a shared memory model. The correctness proof was carried out for an arbitrary, finite number of processors and under the assumption that message order is preserved. The modeling was done through a program written in a guarded command language similar to UNITY, and the proof that the program satisfies its specifications was carried out with Linear-Time Temporal Logic, making use of history variables. Structuring of the proof was achieved by means of auxiliary predicates.

IEEE standards have been the target of study of other formalizations. The IEEE 1394 high performance serial multimedia bus protocol allows several components to communicate with each other at high speed. Devillers, Griffioen, Romijn and Vaandrager [DGRV00] gave a formal model and verification of a leader election algorithm that forms the core of the tree identify phase of the physical layer of the 1394 protocol. The authors first described the algorithm in the I/O Automata model, then they used a standard refinement approach to verify that, for any arbitrary tree topology, exactly one leader is elected. Stoelinga and Vaandrager [SV99] have tackled another aspect of IEEE1394: the root contention protocol. This protocol involves both real time and randomization and it has to be run to find out which node of a tree is root. The verification of the algorithm was carried out in the Probabilistic Automata model of Segala and Lynch [Seg95] [SL95] and the correctness of the protocol was proved by establishing a probabilistic simulation between the implementation and the specification, both Probabilistic Automata.

Most of the formalization work that has been done up to now is about relatively simple algorithms (for example the Alternating Bit or the Leader Election protocols) and/or relatively simple settings (for example absence of process crashes or processes communicating only to their direct neighbors). To our knowledge, there are only few papers dealing with complex algorithms like Consensus and/or in non-trivial settings with failures and unreliable communication.

Pogosyants, Segala and Lynch [PSL00] used Probabilistic I/O Automata as the basis for a formal presentation and proof of the randomized consensus algorithm of Aspnes and Herlihy. They expressed the properties of the system in terms of sequences of external actions that the automata can provide (traces) and they carried out the proofs showing the validity of some invariants on the traces.

De Prisco, Lampson and Lynch [DLL97] also used automata under the form of Clock General Timed Automaton models to give a new presentation of the Paxos algorithm. They decomposed the algorithm into several interacting components, providing a correctness proof, together with a fault tolerance analysis.

Gafni and Lamport [GL03] used TLA+ to describe the algorithm and the proofs of correctness of Disk Paxos, a variant of Paxos based on a network of processes and disks.

In [NFM03] we modeled Chandra and Toueg's algorithm using a CCS-like process calculus. Here, we have reformulated the problem in terms of a rewriting system to alleviate the burden of exhibiting formal proofs. Recently, Francalanza and Hennessy [FH07] have designed a partial-failure process calculus to describe distributed algorithms, and they have proposed a methodology for proving correctness using fault-tolerance bisimulation techniques. More precisely, they have given a bisimulation-based proof for a much simpler Consensus algorithm relying on perfect failure detectors.

Apart from this application-oriented work, Nestmann and Fuzzati [NF03] modeled several classes of failure detectors of [CT96]. Their model was slightly more explicit than the one presented here because it was enforcing failure detector properties on the basis of process' identity (checking whether processes were crashed or elected as "trusted and immortal") and of the actions they tried to perform.

As a final remark, we point out that our notions of Qpp- Rbc-message histories recall that of *history variables*. History variables were mentioned for the first time by Clint in [Cli73] to state and prove correctness criteria of programs and computations. The closest definition to the kind of history variables presented in this paper was given by Abadi and Lamport [AL91]. They used history variables as an augmentation of the state space, in such a way that the additional components recorded past information to be used in the proofs, but did not affect the behavior of the original state components.

Future work. We are interested in studying and comparing the unlabeled transition system of the current paper, based on an explicit representation of the environment, with a labeled transition system that is based on unspecified environments. We believe that the kind of compositional reasoning that one may hope for is not one based on parallel composition across distributed components, but rather one across layers (communication abstractions, failure detectors, and processes). We also believe that our approach can be reused to formalize other distributed algorithms. By simple adaptation of the rules of Tables 2 and 3, we have already given a first formalization of the Paxos Consensus algorithm for crash failures and quasi-reliable channels. We are currently working on introducing recovery of processes and lossy-duplicating channels. Finally, we believe that our formalization of the algorithm together with the correctness arguments represent a solid basis towards a machine-checkable representation.

Acknowledgments. We thank Sergio Mena, André Schiper, Pawel Wojciechowski, Rachid Guerraoui and Roberto Segala for discussions on the Consensus problem; James Leifer, Peter Sewell, Holger Hermanns for discussions on proof techniques. We also wish to thank Philipp Küfner for his careful proof-reading and the anonymous referees for their useful comments.

A Proofs from Section 5

Proof of Lemma 5.2.2

Proof. As to the first three statements, since $\mathbf{Q}_0 = \emptyset$, and since $\mathbf{Q}.X \neq \bot$ (with X equals to $\mathsf{P1}^r_{i\to\mathrm{crd}(r)}$ for (1), $\mathsf{P2}^r_{\mathrm{crd}(r)\to i}$ for (2), $\mathsf{P3}^r_{i\to\mathrm{crd}(r)}$ for (3)) we can infer that entry X must have been added during the run. The only way of adding QPP-messages to the history during a run is by executing a specific transition rule ((PHS-1) for (1), (PHS-2) for (2), (PHS-3-SUSPECT) or (PHS-3-TRUST) for (3)). Note that, even if we suppose that the very same entry can be added more than once per run (message duplication), since $\mathbf{Q}_0 = \emptyset$ we can always identify and choose a point in the run where the entry is introduced for the first time. The resulting data follows from the analysis of the premises and the conclusions of each of those rules.

Few more words need to be spent for the conclusion $\mathrm{val}^r(\mathbf{Q}_{t-1}) \neq \bot$ in 3b, since it does not appear explicitly in the rules. From its definition, we know that $\mathrm{val}^r(\mathbf{Q}_{t-1}) = v$ if for all $j \in \mathbb{P}$ it holds that $\mathbf{Q}_{t-1}.\mathsf{P2}^r_{\mathrm{crd}(r)\to j} = ((v,\cdot),\cdot)$. From the premises of rule (PHS-3-TRUST), we know that $\mathbf{Q}_{t-1}.\mathsf{P2}^r_{\mathrm{crd}(r)\to i} = ((v,s),\mathrm{received})$ for some $i \in \mathbb{P}$. However, all the 2nd-phase messages of a round r are created at the same time, by $\mathrm{crd}(r)$, by performing the rule (PHS-2). As a consequence, for all $j \in \mathbb{P}$, it holds that $\mathbf{Q}_{t-1}.\mathsf{P2}^r_{\mathrm{crd}(r)\to j} = ((v,\cdot),\cdot)$ (even if some j crashed). Thus, $\mathrm{val}^r(\mathbf{Q}_{t-1})$ is defined.

The proof in the broadcast case (4) is completely analogous to the previous ones with respect to the choice of the first addition (out of possibly several) of a broadcast message, but instead using the rule (PHS-4-SUCCESS) to directly read off the relevant data for $\mathbf{B}_t(r)$, $\mathbf{S}_{t-1}(\mathrm{crd}(r))$ and $\mathbf{S}_t(\mathrm{crd}(r))$. □

Proof of Lemma 5.2.3

Proof. The main reasoning is done by contradiction.

1. If (RULE) is (PHS-1), then $\mathbf{S}_{u-1}(i) = \big((r,\mathsf{P1}),\cdot,\cdot\big)$.
   Suppose by contradiction that $\mathbf{Q}_{u-1}.\mathsf{P1}^r_{i\to\mathrm{crd}(r)} \neq \bot$. By an application of Lemma 5.2.2(1) there is $t \in T$, $t \leq u-1$, such that

   $$\langle\, \mathbf{B}_{t-1}, \mathbf{Q}_{t-1}, \mathbf{S}_{t-1} \,\rangle \;\rightarrow_{i:(\text{PHS-1})}\; \langle\, \mathbf{B}_t, \mathbf{Q}_t, \mathbf{S}_t \,\rangle$$

   with $\mathbf{S}_{t-1}(i) = \big((r,\mathsf{P1}),\cdot,\cdot\big)$ and $\mathbf{S}_t(i) = \big((r,P),\cdot,\cdot\big)$, for $P \in \{\mathsf{P2},\mathsf{P3}\}$. Observe that (by Lemma 5.2.1) this data combines into $\mathbf{S}_t(i) = \big((r,P),\cdot,\cdot\big) > \big((r,\mathsf{P1}),\cdot,\cdot\big) = \mathbf{S}_{u-1}(i)$. On the other hand, since $t \leq u-1$ and since process states are non-decreasing (Lemma 5.2.1), we have $\mathbf{S}_t(i) \leq \mathbf{S}_{u-1}(i)$. Thus, the assumption $\mathbf{Q}_{u-1}.\mathsf{P1}^r_{i\to\mathrm{crd}(r)} \neq \bot$ must have been wrong.

2. Completely analogous to the previous case, but using Lemma 5.2.2(2)
3. Analogous to the previous cases, but using Lemma 5.2.2(3a and 3b) and taking into account that one out of two rules may be chosen. In either of the two sub-cases, the data $\mathbf{S}_t(i) = \big((r, \mathsf{P3}), \cdot, \cdot\big)$ that is relevant for the proof is the same.
4. Analogous to the previous cases, but instead using Lemma 5.2.2(4).

□

Proof of Lemma 5.3.2

Proof.

1. Since $(\hat{r}, \mathsf{P1}) <_i (r, P)$, Lemma 5.3.1 provides us with the existence of some $t \in \big[\!\big[0, z\big]\!\big]$ such that $\mathbf{S}_t(i) = ((\hat{r}, \mathsf{P1}), \cdot, \cdot)$. Moreover, since $(\hat{r}, \mathsf{P1}) \neq (r, P)$, there must be a time u with $t < u \leq z$ when process i has left the program counter $(\hat{r}, \mathsf{P1})$ of state $\mathbf{S}_t(i)$ via a transition

$$\langle\, \mathbf{B}_{u-1}, \mathbf{Q}_{u-1}, \mathbf{S}_{u-1} \,\rangle \;\rightarrow_{i:(\textsc{Rule})}\; \langle\, \mathbf{B}_u, \mathbf{Q}_u, \mathbf{S}_u \,\rangle$$

   with $\mathbf{S}_u(i) \neq \mathbf{S}_{u-1}(i) = \mathbf{S}_t(i)$. This transition must be derived by the rule (Rule) = (phs-1), because it is the only one that can change a program counter that involves phase $\mathsf{P1}$. Note that we are not interested in the state $\mathbf{S}_u(i)$, but rather in the fact that the application of this rule necessarily defines an entry for $\mathbf{Q}_u.\mathsf{P1}^{\hat{r}}_{i \to \mathrm{crd}(\hat{r})}$, where $\hat{r}$ is copied from the program counter of $\mathbf{S}_{u-1}(i) = \mathbf{S}_t(i)$.
   From Corollary 5.2.5 we have then $\mathbf{Q}_z.\mathsf{P1}^{\hat{r}}_{i \to \mathrm{crd}(\hat{r})} \neq \bot$.
2. Analogous to the previous case. The only difference is that we have to argue for the definedness of n entries $\mathbf{Q}_u.\mathsf{P2}^{\hat{r}}_{\mathrm{crd}(\hat{r}) \to k}$, for $k \in \mathbb{P}$. We can immediately convince ourselves that this is true by looking at the effects of the execution of rule (phs-2), that is the only rule applicable for a process in phase $\mathsf{P2}$.
3. Again, mostly analogous to the first case. Here, the difference is that there are two rules to be considered for leaving a program counter that mentions phase $\mathsf{P3}$, namely (phs-3-trust) and (phs-3-suspect). By definition, both of these two rules define the entry $\mathbf{Q}_u.\mathsf{P3}^{\hat{r}}_{i \to \mathrm{crd}(\hat{r})}$, and the same proof steps apply as above.

□

Proof of Lemma 5.3.3

Proof. Let us prove first Items 1 and 2 together. We do it by induction on $t \in T$, corresponding to the length of the prefix $\langle\, \mathbf{B}_0, \mathbf{Q}_0, \mathbf{S}_0^{(v_1, \ldots, v_n)} \,\rangle \;\rightarrow^t \langle\, \mathbf{B}_t, \mathbf{Q}_t, \mathbf{S}_t \,\rangle$ of the given run.

Base case $t = 0$ **:** By definition, for all $i \in \mathbb{P}$, we have $\mathbf{S}_0(i) = \big((0, \mathsf{W}), (v_i, 0), \bot\big)$.
Inductive case $t > 0$ **:** We analyze the step

$$\langle\, \mathbf{B}_{t-1}, \mathbf{Q}_{t-1}, \mathbf{S}_{t-1} \,\rangle \;\rightarrow_{j:(\textsc{Rule})}\; \langle\, \mathbf{B}_t, \mathbf{Q}_t, \mathbf{S}_t \,\rangle$$

of the given run, with $\mathbf{S}_{t-1}(i) = \big((r_{t-1}, P_{t-1}), (\cdot, s_{t-1}), \cdot\big)$ and $\mathbf{S}_t(i) = \big((r_t, P_t), (\cdot, s_t), \cdot\big)$
If $i \neq j$, then the desired stamp consistency statement is immediately true by the induction hypothesis, because the last transition only touches the program counter of process j. If $i = j$, then we proceed by case analysis on P_t.

1. If $P_t \in \{\mathsf{P4}, \mathsf{W}\}$ there are two possibilities:
   (a) $P_{t-1} \in \{\mathsf{P4}, \mathsf{W}\}$, and by inductive hypothesis $s_{t-1} \leq r_{t-1}$.
   In this case rule (Rule) can only be:
   - either a rule which does not affect the state of process i, so $s_t = s_{t-1} \leq r_{t-1} = r_t$;
   - or one of the following rules: (phs-4-fail), (phs-4-success), (Rbc-deliver). Also in this case it is $s_t = s_{t-1} \leq r_{t-1} = r_t$.

(b) $P_{t-1} = \mathsf{P3}$, and by inductive hypothesis $s_{t-1} < r_{t-1}$.
In this case rule (RULE) can only be:
- either rule (PHS-3-SUSPECT), and $s_t = s_{t-1} < r_{t-1} = r_t$;
- or rule (PHS-3-TRUST), and $s_t = r_t$.

Note that we do not consider $P_{t-1} \in \{\mathsf{P1}, \mathsf{P2}\}$ since it is not possible from those phases to reach $P_t \in \{\mathsf{P4}, \mathsf{W}\}$ with just one transition.

2. If $P_t \in \{\mathsf{P1}, \mathsf{P2}, \mathsf{P3}\}$ there are two possibilities:
   (a) $P_{t-1} = \mathsf{W}$, and by inductive hypothesis $s_{t-1} \leq r_{t-1}$.
   Then, the only applicable (RULE) is (WHILE), for which $s_t = s_{t-1} \leq r_{t-1} < r_{t-1}+1 = r_t$.
   Note that we do not consider $P_{t-1} = \mathsf{P4}$ since it is not possible from that phase to reach $P_t \in \{\mathsf{P1}, \mathsf{P2}, \mathsf{P3}\}$ with just one transition.
   (b) $P_{t-1} \in \{\mathsf{P1}, \mathsf{P2}, \mathsf{P3}\}$, and by inductive hypothesis $s_{t-1} < r_{t-1}$.
   In this case rule (RULE) can only be:
   - either a rule which does not affect the state of process i, so $s_t = s_{t-1} < r_{t-1} = r_t$;
   - or one of the rules (PHS-1), (PHS-2), (RBC-DELIVER), where $s_t = s_{t-1} < r_{t-1} = r_t$.

Let us prove now the third and last statement. If $t = z$ then $s_t = s_z$. If $t < z$ then we will prove that for all $w \in T$, with $t \leq w < z$, it holds that $s_w \leq s_{w+1}$. Thanks to the transitivity properties of $<$, we can prove our claim through an iteration of this result for all w.

Let us fix $w \in T$ such that $t \leq w < z$. Now, consider the transition

$$\langle\, \mathbf{B}_w, \mathbf{Q}_w, \mathbf{S}_w \,\rangle \;\rightarrow_{j:(\text{RULE})}\; \langle\, \mathbf{B}_{w+1}, \mathbf{Q}_{w+1}, \mathbf{S}_{w+1} \,\rangle$$

for $j \in \mathbb{P}$. If $j \neq i$ then $s_w = s_{w+1}$. Suppose now $j = i$. By inspection on the transition rules of Tables 2 and 3:

- If (RULE) is any rule but (PHS-3-TRUST) then the belief component of the state remains unchanged, and hence $s_w = s_{w+1}$.
- If (RULE)=(PHS-3-TRUST) then process i evolves into phase $\mathsf{P4}$ and adopts the round proposal with the current round number as stamp, and hence $s_{w+1} = r_w$. By Item 1 $s_w \leq r_w$, hence $s_w \leq s_{w+1}$.

□

Proof of Lemma 5.3.6

Proof. As $\mathbf{Q}_z.\mathsf{P1}^r_{i\rightarrow \mathrm{crd}(r)} = ((v,s),\cdot)$, by Lemma 5.2.2(1) there is $u \in T$, $u \leq z$, for which

$$\langle\, \mathbf{B}_{u-1}, \mathbf{Q}_{u-1}, \mathbf{S}_{u-1} \,\rangle \;\rightarrow_{i:(\text{PHS-1})}\; \langle\, \mathbf{B}_u, \mathbf{Q}_u, \mathbf{S}_u \,\rangle$$

with $\mathbf{S}_{u-1}(i) = \big((r, \mathsf{P1}), (\cdot, s), \cdot\big)$ and $\mathbf{S}_u(i) = \big((r, \cdot), (\cdot, s), \cdot\big)$.
As $\mathbf{Q}_z.\mathsf{P1}^{r'}_{i\rightarrow \mathrm{crd}(r')} = ((v', s'), \cdot)$, $\mathbf{Q}_z.\mathsf{P1}^r_{i\rightarrow \mathrm{crd}(r)} = ((v,s),\cdot)$, and $0 < r' < r$, by Lemma 5.3.4(2), for all rounds r'', $r' \leq r'' < r$, there is a stamp s'' such that $\mathbf{Q}_z.\mathsf{P1}^{r''}_{i\rightarrow \mathrm{crd}(r'')} = \big((\cdot, s''), \cdot\big)$, with $s' \leq s'' \leq s$.
Whenever $r'' < r$ then $(r'', \mathsf{P3}) <_i (r, \mathsf{P1})$, and by Lemma 5.3.2(3) it holds that $\mathbf{Q}_{u-1}.\mathsf{P3}^{r''}_{i\rightarrow \mathrm{crd}(r'')} \neq \perp$.
By Corollary 5.2.5(1), it follows that $\mathbf{Q}_z.\mathsf{P3}^{r''}_{i\rightarrow \mathrm{crd}(r'')} \neq \perp$, for all r'', with $r' \leq r'' < r$.

1. Let $s' = s$.
   Then for all rounds r'', with $r' \leq r'' < r$, we have $\mathbf{Q}_z.\mathsf{P1}^{r''}_{i\rightarrow \mathrm{crd}(r'')} = \big((\cdot, s), \cdot\big)$, and since $r' \leq r'' < r''+1 \leq r$ and $\mathbf{Q}_z.\mathsf{P1}^r_{i\rightarrow \mathrm{crd}(r)} = \big((\cdot, s), \cdot\big)$ we also have $\mathbf{Q}_z.\mathsf{P1}^{r''+1}_{i\rightarrow \mathrm{crd}(r''+1)} = \big((\cdot, s), \cdot\big)$.
   Now, suppose by contradiction that that there is a round $\hat{r}$, with $r' \leq \hat{r} < r$, such that $\mathbf{Q}_z.\mathsf{P3}^{\hat{r}}_{i\rightarrow \mathrm{crd}(\hat{r})} = (\mathrm{ack}, \cdot)$.

As $\mathbf{Q}_z.\mathsf{P1}^{\hat{r}}_{i\to\mathrm{crd}(\hat{r})}=\big((\cdot,s),\cdot\big)$ and $\mathbf{Q}_z.\mathsf{P1}^{\hat{r}+1}_{i\to\mathrm{crd}(\hat{r}+1)}=\big((\cdot,s),\cdot\big)$, by Lemma 5.3.5(2) we would obtain $s < s$. Thus, it must be $\mathbf{Q}_z.\mathsf{P3}^{r''}_{i\to\mathrm{crd}(r'')}=(\mathrm{nack},\cdot)$ for all rounds r'', with $r'\leq r''<r$. By applying Lemma 5.3.5(1) to all rounds r'', we derive $v' = v$.

2. Let $s' < s$.
   Suppose by contradiction that for all r'', $r'\leq r''<r$, it holds that $\mathbf{Q}_z.\mathsf{P3}^{r''}_{i\to\mathrm{crd}(r'')}=(\mathrm{nack},\cdot)$. Then, by applying Lemma 5.3.5(1) to all rounds r'', with $r'\leq r''<r$, we derive $s' = s$. So, there must be $\hat{r}$, with $r'\leq\hat{r}<r$, such that $\mathbf{Q}_z.\mathsf{P3}^{\hat{r}}_{i\to\mathrm{crd}(\hat{r})}=(\mathrm{ack},\cdot)$.

□

References

[ACT97] Marcos Kawazoe Aguilera, Wei Chen, and Sam Toueg. Heartbeat: A timeout-free failure detector for quiescent reliable communication. In Marios Mavronicolas and Philippas Tsigas, editors, *Proceedings of the 11th International Workshop on Distributed Algorithms*, volume 1320 of *Lecture Notes in Computer Science*, pages 126–140. Springer-Verlag, 1997.

[AL91] M. Abadi and L. Lamport. The existence of refinement mappings. *Theoretical Computer Science*, 82(2):253–284, 1991. A preliminary version appeared in the proceedings of LICS'88.

[CHT96] Tushar Deepak Chandra, Vassos Hadzilacos, and Sam Toueg. The weakest failure detector for solving consensus. *Journal of the ACM*, 43(4):685–722, 1996.

[Cli73] Maurice Clint. Program proving: Coroutines. *Acta Informatica*, 2:50–63, 1973.

[CM88] K. M. Chandy and J. Misra. *Parallel Programming Design. A Foundation.* Addison Wesley, reading MA, 1988.

[CT96] Tushar Deepak Chandra and Sam Toueg. Unreliable failure detectors for reliable distributed systems. *Journal of the ACM*, 43(2):225–267, 1996.

[DGRV00] M. Devillers, W. O. D. Griffioen, J. Romijn, and F. W. Vaandrager. Verification of a leader election protocol: Formal methods applied to ieee 1394. *Formal Methods in System Design*, 16(3):307–320, 2000.

[DLL97] Roberto De Prisco, Butler Lampson, and Nancy Lynch. Revisiting the paxos algorithm. In M. Mavronicolas and Ph. Tsigas, editors, *Distributed Algorithms; Proceedings of the 11th International Workshop, WDAG'97, Saarbrcken, Germany*, volume 1320 of *Lecture Notes in Computer Science*, pages 111–125. Springer-Verlag, 1997. September.

[FGK97] L. Fredlund, J. F. Groote, and H. Korver. Formal verification of a leader election protocol in process algebra. *Theoretical Computer Science*, 177(2):459–486, 1997.

[FH07] A. Francalanza and M. Hennessy. A fault tolerance bisimulation proof for consensus. In *Proceedings of the 16th European Symposium on Programming ESOP '07*. Springer-Verlag, 2007.

[FLP85] Michael J. Fischer, Nancy A. Lynch, and Michael S. Paterson. Impossibility of distributed consensus with one faulty process. *Journal of the ACM*, 32(2):374–382, 1985.

[FS96] A. Felty and F. Stomp. A correctness proof of a cache coherence protocol. In *Compass'96: Eleventh Annual Conference on Computer Assurance*, page 128, Gaithersburg, Maryland, 1996. National Institute of Standards and Technology.

[GL03] Eli Gafni and Leslie Lamport. Disk paxos. *Distributed Computing*, 16(1):1–20, 2003.

[GP91] J. F. Groote and A. Ponse. Proof theory for μCRL. Technical Report CS-R9138, Amsterdam, 1991.

[GS01] J. F. Groote and J. Springintveld. Focus points and convergent process operators: a proof strategy for protocol verification. *J. Log. Algebr. Program.*, 49(1-2):31–60, 2001.

[Jon87] B. Jonsson. *Compositional Verification of Distributed Systems.* PhD thesis, Department of Computer Systems, Uppsala University, 1987.

[Lam02] L. Lamport. *Specifying Systems: The TLA+ Language and Tools for Hardware and Software Engineers.* Pearson Education, 2002.

[LT87] N. A. Lynch and M. R. Tuttle. Hierarchical correctness proofs for distributed algorithms. In *PODC*, pages 137–151, 1987.

[LV95] N. A. Lynch and F. W. Vaandrager. Forward and backward simulations – part I: untimed systems. *Information and Computation*, 121(2):214–233, 1995.

[NF03] Uwe Nestmann and Rachele Fuzzati. Unreliable failure detectors via operational semantics. In Vijay A. Saraswat, editor, *Proceedings of ASIAN 2003*, volume 2896 of *Lecture Notes in Computer Science*, pages 54–71. Springer-Verlag, December 2003.

[NFM03] Uwe Nestmann, Rachele Fuzzati, and Massimo Merro. Modeling consensus in a process calculus. In Roberto Amadio and Denis Lugiez, editors, *Proceedings of CONCUR 2003*, volume 2761 of *Lecture Notes in Computer Science*, pages 399–414. Springer-Verlag, August 2003.

[PSL00] A. Pogosyants, R. Segala, and N. Lynch. Verification of the randomized consensus algorithm of aspnes and herlihy: a case study. *Distributed Computing*, 13(3):155–186, 2000.

[Seg95] R. Segala. *Modeling and Verification of Randomized Distributed Real-Time Systems.* PhD thesis, Laboratory for Computer Science, Massachusetts Institute of Technology, 1995.

[SL95] R. Segala and N. Lynch. Probabilistic simulations for probabilistic processes. *Nordic Journal of Computing*, 2(2):250–273, 1995.

[SV99] M. Stoelinga and F. Vaandrager. Root contention in IEEE 1394. *Lecture Notes in Computer Science*, 1601:53–74, 1999.

[WSS97] S. Wu, S. A. Smolka, and E. W. Stark. Composition and behaviors of probabilistic I/O automata. *Theoretical Computer Science*, 176(1–2):1–38, 1997.